

**INTERNATIONAL ORGANISATION FOR STANDARDISATION  
ORGANISATION INTERNATIONALE DE NORMALISATION  
ISO/IEC JTC1/SC29/WG11  
CODING OF MOVING PICTURES AND ASSOCIATED AUDIO  
INFORMATION**

**ISO/IEC JTC1/SC29/WG11**

**N1277**

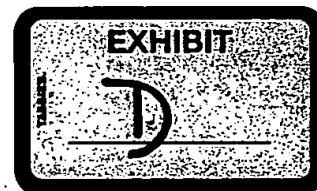
**Tampere, July 96**

**Source: Video Group**

**Status: Draft in Progress**

**Title: MPEG-4 Video Verification Model Version 3.0**

**Author: Ad hoc group on MPEG-4 video VM editing**



Please address all comments or suggestions to the ad hoc group on MPEG-4 video VM editing mail reflector 'mpeg4-vm@ltssg3.epfl.ch'.

<b>1. Introduction</b>	<b>4</b>
<b>2. Video Object Plane (VOP)</b>	<b>4</b>
2.1 VOP Definition	4
2.2 VOP format	5
<b>3. Encoder Definition</b>	<b>9</b>
3.1 Overview	9
3.2 Shape Coding	10
3.2.1.5.1 Framework	19
3.2.1.5.2 Description	19
3.3 Motion Estimation and Compensation	25
3.4 Texture Coding	35
3.5 Prediction and Coding of B-VOPs	41
3.6 Rate Control	44
3.7 Generalized Scalable Encoding	44
<b>4. Bitstream Syntax</b>	<b>50</b>
4.1 General Structure	50
4.2 Video Session Class	51
4.3 Video Object Class	52
4.4 Video Object Layer Class	53
4.5 VideoObjectPlane Class	56
4.6 Shape coding	59
4.7 Motion Shape Texture	62
<b>5. Decoder Definition</b>	<b>64</b>
5.1 Overview	64
5.2 Shape decoding	64
5.3 Generalized Scalable Decoding	65
5.4 Compositor Definition	69
5.5 Flex_0 Composition Layer Syntax	69
<b>Appendix A</b>	<b>72</b>
<b>Combined Motion Shape Texture Coding</b>	<b>72</b>
B.1 Macroblock Layer	72
B.2 Block Layer	82
<b>Appendix B</b>	<b>86</b>
<b>Core Experiments</b>	<b>86</b>

## 1. Introduction

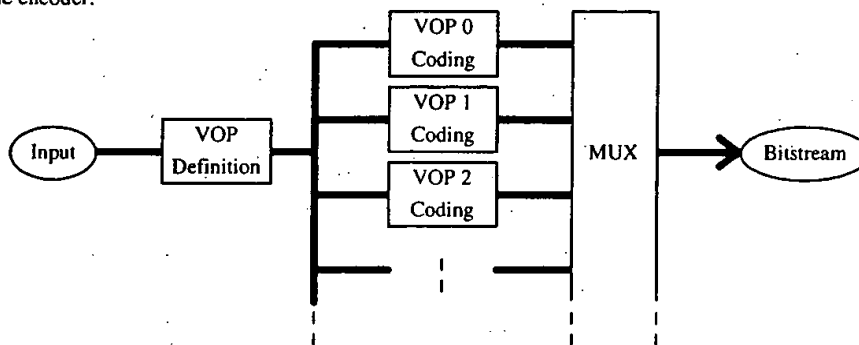
tbd

## 2. Video Object Plane (VOP)

### 2.1 VOP Definition

The Video Object Planes (VOP) correspond to entities in the bitstream that the user can access and manipulate (cut, paste...). The encoder sends together with the VOP, composition information to indicate where and when each VOP is to be displayed. At the decoder side the user may be allowed to change the composition of the scene displayed by interacting on the composition information.

At the encoder:



At the decoder:

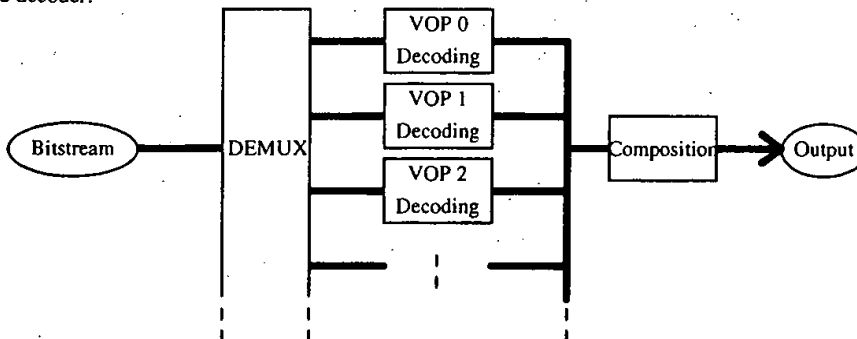


Figure 2.1.1: VM Encoder and Decoder Structure

The VOP can be a semantic object in the scene : it is made of Y, U, V components plus shape information. In MPEG-4 video test sequences, the VOP were either known by construction of the sequences (hybrid

sequences based on blue screen composition or synthetic sequences) or were defined by semi-automatic segmentation. In the first case, the shape information is represented by an 8 bit component, used for composition (see section 5.2). In the second case, the shape is a binary mask. Both cases are currently considered in the encoding process. The VOP can have arbitrary shape.

The exact method used to produce the VOP from the video sequences is not described in this document.

When the sequence has only one rectangular VOP of fixed size displayed at fixed interval, it corresponds to the frame-based coding technique.

## **2.2 VOP format**

This section describes the input library, the filtering process and the formation of the VOP.

Section 2.2.1 describes the test sequences library. Section 2.2.2 describes the suggested downsampling process from ITU-R 601 format to SIF, CIF and QCIF formats. In this section, the acronym SIF is used to designate the 352x240 and 352x288 formats at 30 Hz and 25 Hz, respectively, while CIF designates only the 352x288 format at 30 Hz. Section 2.2.3 describes the VOP format.

### **2.2.1 Test sequences library**

All the test sequences will be available in either 50 Hz or 60 Hz ITU-R 601 formats. The input library from the November '95 and January '96 test was adopted here. As the VM evolves it is expected that more representative sets of input source will become available. The distributed files format for the input sources are as follows:

- 1) **Luminance and chrominance (YUV) - ITU-R 601 format** containing luminance and chrominance data
  - one or more file per sequence;
  - no headers
  - supply number of files and size in separate README file
  - chain all frames without gaps
  - for each frame, chain Y, U, V data without gaps
  - write component data from 1st line, 1st pixel, from left to right, top to bottom, down to last line, last pixel.
- 2) **Segmentation Masks - The format for the exchange of the mask information is similar to the one used for the images, i.e. a segmentation mask has a format similar to ITU-R 601 luminance, where each pixel has a label identifying the region it belongs to (label values are 0,1,2, ...). A segmentation may have a maximum of 256 segments (regions). Whenever possible, the segments should have a semantic meaning and will correspond to the VOP.**
- 3) **Grey Scale Alpha Plane files - ITU-R 601 format - containing the alpha values. The same format as the ITU-R 601 luminance file is used. All values between 0 and 255 may be used. For the layered representation of a sequence, each layer has its own YUV and alpha files.**

The test sequences library is separated into the following classes:

- Class A: Low spatial detail and low amount of movement
- Class B: Medium spatial detail and low amount of movement or vice versa
- Class C: High spatial detail and medium amount of movement or vice versa

Class D: Stereoscopic  
Class E: Hybrid natural and synthetic

The following table lists the input sequences, their format and the available files.

Sequence Name	Class	Input Format	YUV files	Alpha files	Segment Mask Available
Mother & daughter	A	ITU-R 601 (60Hz)	1	0	0
Akiyo	A	ITU-R 601 (60Hz)	2+1	1	2
Hall Monitor	A	ITU-R 601 (60Hz)	1	0	3
Container Ship	A	ITU-R 601 (60Hz)	1	0	6
Sean	A	ITU-R 601 (60Hz)	1	0	3
Foreman	B	ITU-R 601 (50Hz)	1	0	0
News	B	ITU-R 601 (60Hz)	4+1	3	4
Silent Voice	B	ITU-R 601 (50Hz)	1	0	0
Coastguard	B	ITU-R 601 (60Hz)	1	0	4
Bus	C	ITU-R 601 (60Hz)	1	0	0
Table Tennis	C	ITU-R 601 (50Hz)	1	0	0
Stefan	C	ITU-R 601 (60Hz)	1	0	2
Mobile & Calendar	C	ITU-R 601 (60Hz)	1	0	0
Tunnel	D	ITU-R 601 (50Hz)	2x1	0	0
Fun Fair	D	ITU-R 601 (50Hz)	2x1	0	0
Children	E	ITU-R 601 (60Hz)	3+1	2	3
Bream	E	ITU-R 601 (60Hz)	3+1	2	3
Weather	E	ITU-R 601 (60Hz)	2+1	1	2
Destruction	E	ITU-R 601 (60Hz)	11+1	10	0

Table 2.2.1 Lists of input library files

Note: N+1 indicates that the sequence consists of N layers and the composed sequence.  
Nx1 indicates the the sequences consists of N views.

### 2.2.2 Filtering process

The filtering process for YUV is based on the document [MPEG95/0322]. The filtering process for alpha planes (A) is based on the document [MPEG95/0393]. Software for performing the filtering process was distributed and can also be obtained from MPEGftp site 'drop.chips.ibm.com:Tampere/Contrib/m0896.zip'.

In the first step, the first field of a picture is omitted (both luminance and chrominance). Then the physically centred 704x240/288 and 352x240/288 pixels are extracted. This format is used to create all the smaller formats using the filters listed in table 2.2.2 and following the steps described below.

	Factor	Tap no.	Filter taps	Divisor
A	1/2	1	5,11,11,5	32
B	1/2	1	2,0,-4,-3,5,19,26,19,5,-3,-4,0,2	64
C	1/4	1	-5,-4,0,5,12,19,24,26,24,19,12,5,0,-4,-5	128
D	6/5	1	-16,22,116,22,-16	128
		2	-23,40,110,1	128
		3	-24,63,100,-11	128
		4	-20,84,84,-20	128
		5	-11,100,63,-24	128
		6	1,110,40,-23	128
E	3/5	1	-24,-9,88,146,88,-9,-24	256
		2	-28,17,118,137,53,-26,-15	256
		3	-15,-26,53,137,118,17,-28	256
F	1/2	1	-12,0,140,256,140,0,-12	512

Table 2.2.2: Filter taps for downsampling

**ITU-R 601 to CIF / SIF**

For Y

704x240 - B -> 352x240 - D -> 352x288

704x288 - B -> 352x288

For U and V

352x240 - B -> 176x240 - D -> 176x288 - A -> 176x144

352x288 - B -> 176x288 - A -> 176x144

For A

704x240 - F -> 352x240 - D -> 352x288

704x288 - F -> 352x288

**ITU-R 601 to QCIF**

For Y and A

704x240 - C -> 176x240 - E -> 176x144

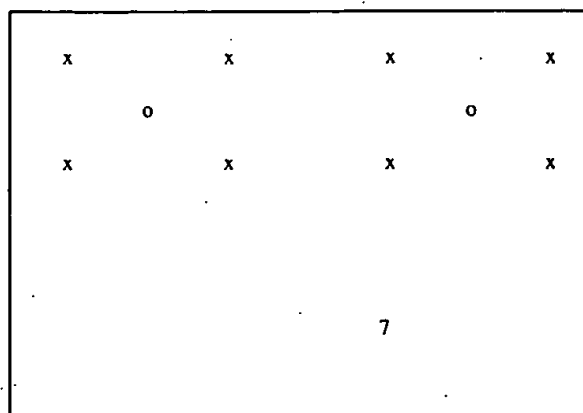
704x288 - C -> 176x288 - B -> 176x144

For U and V

352x240 - C -> 88x240 - E -> 88x144 - A -> 88x72

352x288 - C -> 88x288 - B -> 88x144 - A -> 88x72

The resulting position of the chrominance relative to the luminance is as follows :





where x : luminance, o : chrominance

**Figure 2.2.1 Position of chrominance samples after filtering**

**Notes:**

The 4:2:2 to 4:2:0 conversion is done in the last step because then the correct position of the chroma samples can be preserved.

For input sequences in 4:2:0 format a conversion from 4:2:0 to 4:2:2 is performed before the filtering process starts. The interpolation filter is (1,3,3,1) as specified in document [WG11/N0999].

Filtering of border pixels: When some of the filter taps fall outside the active picture area then the edge pixel is repeated into the blanking area.

**Processing of grey scale alpha planes**

The downsampling process for alpha planes is the same as for luminance (Y). However, for alpha planes a different filter is used for horizontal 2-to-1 filtering. This filter preserves more the high frequency band and therefore maintains a sharp edge for alpha planes.

For the grey scale alpha planes in Class E sequences all the values below a certain threshold are set to 0. The following threshold values are recommended:

Sequence	VOP	Name	Threshold
children	VOP0	children_0	-
	VOP1	children_1	64
	VOP2	children_2	64
weather	VOP0	weather_0	-
	VOP1	weather_1	64
brear	VOP0	brear_0	-
	VOP1	brear_1	64
	VOP2	brear_2	64
destruction	VOP0	destruction_0	-
	VOP1	destruction_1	64
	VOP2	destruction_2	64
	VOP3	destruction_3	64
	VOP4	destruction_4	64
	VOP5	destruction_5	64
	VOP6	destruction_6	64



	VOP7	destruction_7	64
	VOP8	destruction_8	32
	VOP9	destruction_9	64
	VOP10	destruction_10	64

Table 2.2.3. Threshold values for Class E sequences

### Processing of segmentation mask

The segmentation mask is first converted to binary alpha planes.

An object can occupy one or more segments in the segmentation mask. The binary shape information is set to '255' for all pixels that have the label values of the selected segments. All other pixels are considered outside the object and are given a value of '0'.

The downsampling process for the binary alpha plane follows that of the grey scale alpha planes. A threshold of 128 is selected. All filtered values below this threshold are set to '0', whereas all filtered values above the threshold are set to '255'.

### **2.2.3. VOP file format**

The following is the VOP file format. Each VOP consists of a down sampled Y,U and V data file and the alpha plane as specified in section 2.2.2. For simplicity the same alpha file format is used for binary as well as grey scale shape information. For binary shape information the value of 0 is used to indicate a pixel outside of the object and the value of 255 is used to indicate a pixel inside the object. For grey scale shape information the whole range of values between 0 and 255 is used. VOP0 is a special case where the alpha values are all 255. The blending operation is described in section 5.2.

### **2.2.4. Coding of test sequences whose width and height are not integral multiples of 16**

In order to code test sequences whose width and height are not integer multiples of 16 (macroblock size), the width and height of these sequences are first extended to be the smallest integral multiples of 16. The extended areas of the images are then padded using a repetitive padding technique described in 3.3.1.

## **3. Encoder Definition**

### **3.1 Overview**

The Figure 3.1.1 presents a general overview of the VOP encoder structure. The same encoding scheme is applied when coding all the VOPs of a given session.

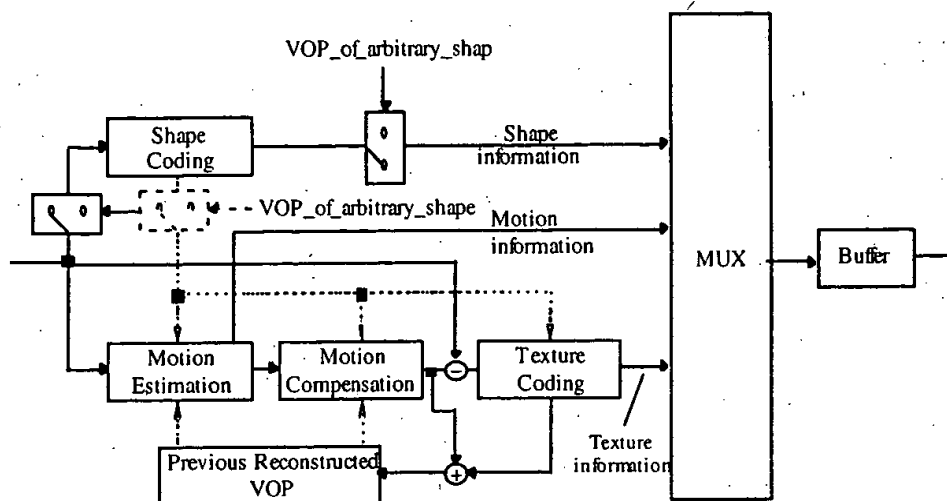


Figure 3.1.1 : VOP encoder structure.

The encoder is mainly composed of two parts : the shape coder and the traditional motion & texture coder applied to the same VOP. The VOP is represented by means of a bounding rectangle as described further. The phase between the luminance and chrominance samples of the bounding rectangle has to be correctly set according to the 4:2:0 format, as shown in Figure 3.1.2. Specifically the top left coordinate of the bounding rectangle should be rounded to the nearest even number not greater than the top left coordinates of the tightest rectangle. Accordingly, the top left coordinate of the bounding rectangle in the chrominance component is that of the luminance divided by two.

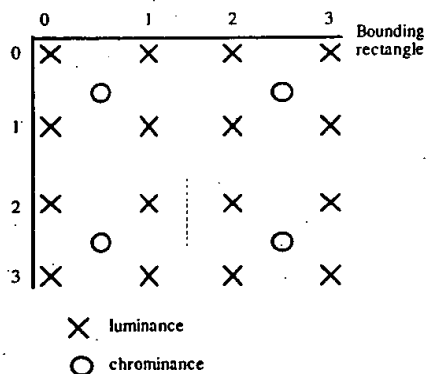


Figure 3.1.2 : Luminance versus chrominance bounding box positioning

### 3.2 Shape Coding

This section describes the coding methods for binary and grey scale shape information. The shape information is hereafter referred to as alpha planes. Binary alpha planes are encoded by modified MMR

while grey scale alpha planes are encoded by quadtree with vector quantization. An alpha plane is bounded by a rectangle that includes the shape of a VOP, as described in section 3.1. The bounding rectangle of the VOP is then extended on the right-bottom side to multiples of 16x16 blocks. The extended alpha samples are set to zero. The extended alpha plane is partitioned into blocks of 16x16 samples (hereafter referred to as alpha blocks) and the encoding/decoding process is done per alpha block.

If the pixels in a macroblock are all transparent (all zero), the macroblock is skipped before motion and/or texture coding. No overhead is required to indicate this mode since this transparency information can be obtained from shape coding. This skipping applies to all I-, P-, and B-VOP's.

### **3.2.1 Binary Shape Coding-Modified MMR**

#### **3.2.1.1 Overview**

- (1) The bounding rectangle of the VOP is extended on the right-bottom side to multiples of 16x16 blocks. The position and the size of the rectangle are coded by VOP\_horizontal\_mc\_spatial\_ref, VOP\_vertical\_mc\_spatial\_ref, VOP\_width, and VOP\_height.
- (2) Basically, MMR coding is disposed by detecting the color-changing pixel (from black(Object) to white(Background) or from white to black), and calculating the distance between the current changing pixel and the previous changing pixel.
- (3) If all pixels in a MB are white (=All white MB) or black (=All black MB), MMR coding is not carried out.
- (4) Fig.M1 shows an example.

(4-1) The color of the first pixel (the top-left pixel) in a MB (=a0\_color) is coded using 1bit (black:a0\_color=1, white:a0\_color=0).

(4-2) If the changing pixel is in the top line in a MB, the bottom line of the reconstructed upper MB is used as a reference area (Top reference in Fig.M1), and if the changing pixel is in the left-end column in a MB, the right-end column of the reconstructed previous MB is used as a reference area (Left reference in Fig.M1). In other cases, the reference area is defined as previous N pixels (N means the number of pixels in one pixel line of a MB) including the changing pixel (see Fig.M2(b)). Therefore, black dots (•) in Fig.M1 are regarded as changing pixels.

(4-3) If a MB is in a top MB line or in a left-side MB column in a frame, the colors of Top reference or Left reference are conveniently set to "white".

(4-4) The distance between the current changing pixel and the previous changing pixel is coded. There are three modes in this algorithm (refer 3.2.1.3 for detail) : Horizontal mode, Vertical Pass mode, and Vertical mode.

(4-5) Shape information of the original size can be size-converted for rate control and rate reduction. The conversion ratios (CR) proposed here are 1 (original size), 1/2, and 1/4. The CR information is VLC coded (see Table M1).

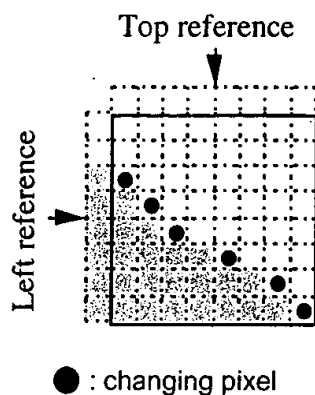


Fig.M1 Changing pixel and reference area

### 3.2.1.2 Detailed explanation

Fig.M2 shows an example of proposed modified MMR.

a0: The starting pixel of the coding disposition.

a1: The next changing pixel to a0.

b1: If b1 is on the r1 of the reference area (see Fig.M2(b)), it means that b1 is the first color-changing pixel and the color is opposed to a0. If b1 is on the r2 of the reference area (see Fig.M2(b)), it means that b1 is the first color-changing pixel from Left reference.

The relative addresses of a0, a1, and b1 can be calculated as follows. Here r\_X means the relative address of pixel X (X=a0, a1, b1), and abs\_X means the absolute address of pixel X counting from the top left of a MB. Furthermore, WIDTH means the number of pixels in one pixel line of a MB.

$$(1) \quad r_{a0} = \text{abs\_a0} - (\text{int})(\text{abs\_a0} / \text{WIDTH}) \times \text{WIDTH}$$

$$(2) \quad r_{a1} = \text{abs\_a1} - (\text{int})(\text{abs\_a1} / \text{WIDTH}) \times \text{WIDTH}$$

Detailed detection algorithm for a1 is described in section 3.2.1.4

$$(3) \quad r_{b1} = \text{abs\_b1} - ((\text{int})(\text{abs\_a0} / \text{WIDTH}) - 1) \times \text{WIDTH}$$

Detailed detection algorithm for b1 is described in section 3.2.1.4

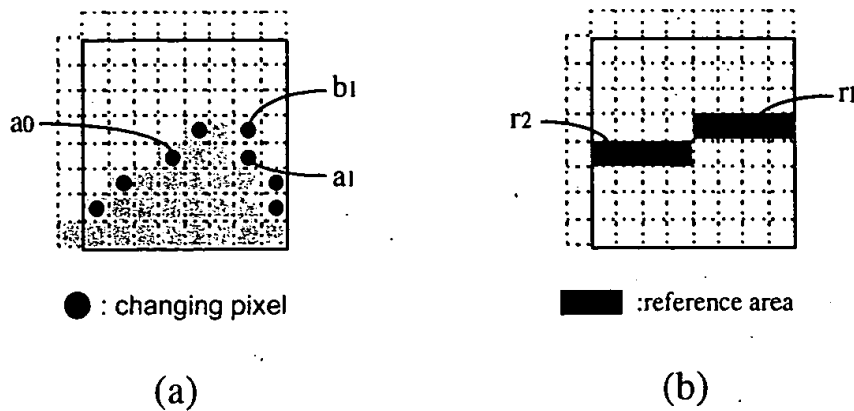


Fig.M2 Modified MMR

### 3.2.1.3 Three modes of modified MMR

#### 3.2.1.3.1 Horizontal mode

Fig.M3 (a) shows an example of this mode.

- In this case, the information of ULB(Unchanged-length of binary information =  $\text{abs\_a1} - \text{abs\_a0}$  : here, the range of ULB is set from 1 to WIDTH) is coded. Actually, the value of (ULB - 1) is coded using fixed-length code word (if WIDTH is 16, four bits are used, and if WIDTH is 8 or 4, three or two bits are used, respectively). Therefore, no table is needed.
- Before Horizontal mode starts, Horizontal mode flag H(=001 : see Table M2) is sent first.
- After the flag H, ULB code is sent.
- Since maximum value of Unchanged-length is WIDTH, Vertical pass mode is used if Unchanged-length is longer than WIDTH (see next section and Fig.M4).

#### 3.2.1.3.2 Vertical Pass mode

Vertical Pass mode is used when the Unchanged-length becomes longer than WIDTH.

- Fig.M4 shows an example of output code words in the case of Vertical Pass mode. From a0 to c0 in Fig.M4 (a), the colors of pixels are the same ("white" in this case). Therefore, this part becomes Horizontal mode (H), and Unchanged-length of binary information(ULB) is WIDTH. But since the colors of pixels in the rest part of this pixel line are also "white", Vertical Pass mode can be used.
- The code word WIDTH is used as the changing signal from Horizontal mode to Vertical Pass mode (Vertical\_pass\_mode flag is set to TRUE). Usually, V0 means DIST=0 of Vertical mode (see Table M2), but V0s after WIDTH mean "This line is Vertical Pass mode".

- In the same way, next two pixel lines (the third and the fourth pixel lines in Fig.M4(a)) are coded as Vertical Pass mode. Even if the color of the line is changed as shown in Fig.M4 (a) (the third line), this line also becomes Vertical Pass mode because there is no change from Left reference.
- The next line (the fifth line) before a1 becomes Horizontal mode again. Therefore,  $abs\_a1-abs\_c1$  is coded using Horizontal mode (c1 means the first pixel of this line). This information is called RLB (=Residual-length of binary information), and the range is set from 0 to WIDTH-1. In this case, RLB code is sent after the flag H, and Vertical\_pass\_mode is reset (=FALSE).
- Fig.M4(b) is an example that Vertical Pass Mode can be used from the beginning of a MB. That is, in the case that a0 is the first pixel of a MB and there is no b1 in the reference area, the mode becomes Vertical Pass mode (Vertical\_pass\_mode=TRUE). To the contrary, if b1 is found in the reference area, the mode becomes Horizontal mode or Vertical mode (Vertical\_pass\_mode=FALSE).

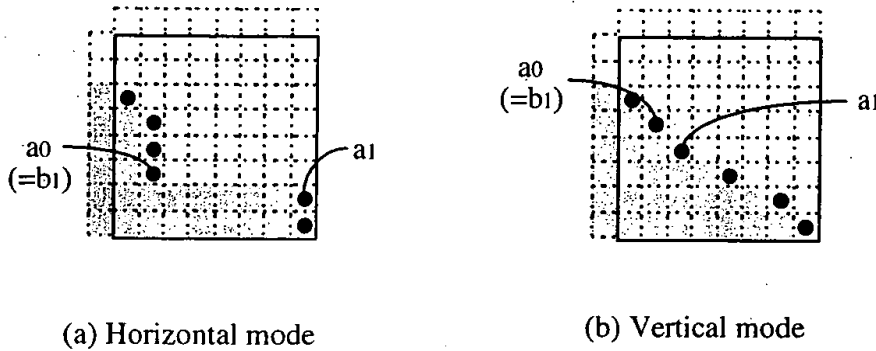
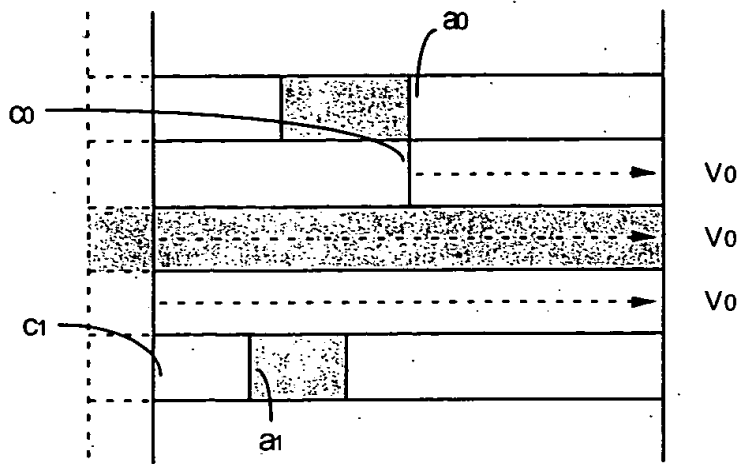
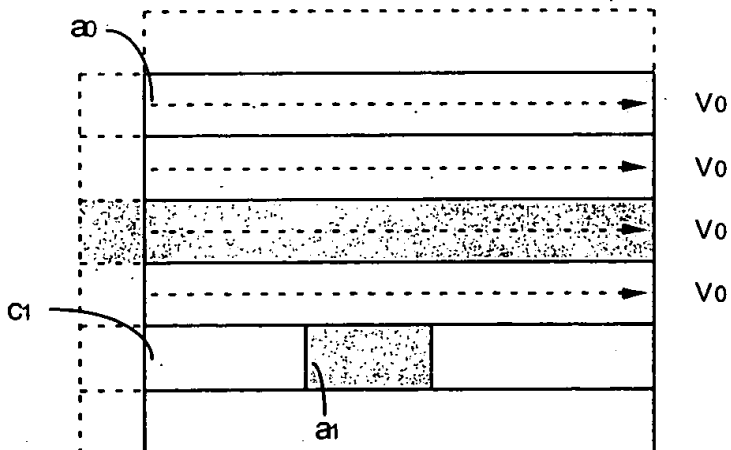


Fig.M3 Horizontal mode and Vertical mode



$$H + \boxed{\text{ULB (WIDTH)}} + V_0 + V_0 + V_0 + H + \boxed{\text{RLB}}$$

(a)



$$V_0 + V_0 + V_0 + V_0 + H + \boxed{\text{RLB}}$$

(b)

Fig.M4 Vertical Pass mode

#### 3.2.1.3.3 Vertical mode

Fig.M3 (b) shows an example of this mode. When this mode is identified, the position of a1 is coded relative to the position of b1 (absolute value of  $r_{a1} - r_{b1} = \text{DIST}$  in Table M2).

#### 3.2.1.3.4 End of MB

After the last change of color, End of MB code (EOMB = '0001' : see Table M2) is sent, and the coding disposition for the MB is finished.

#### 3.2.1.4 Flowchart of encoder

Fig.M5 shows the flow chart of the coding algorithm.



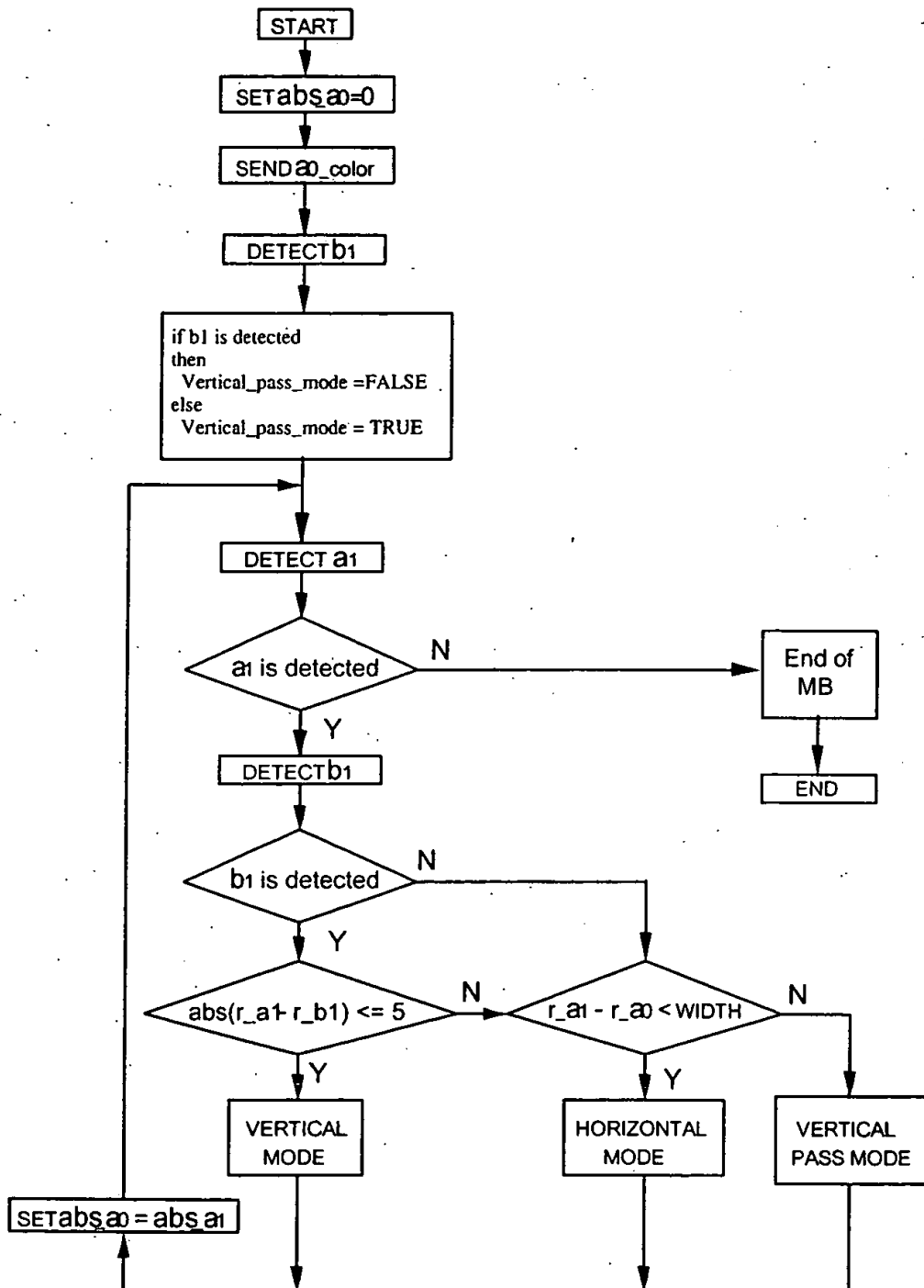


Fig.M5 Flowchart of encoder

The detailed processes of "DETECT a1" and "DETECT b1" in Fig.M5 are as follows (C program):

**(1) DETECT a1**

```
int i, line, a0_line;
i = abs_a0 + 1;
a0_line = abs_a0/WIDTH;
if (i >= total_number_of_pixels) {
    i = -1;
} else {
    do {
        line = i/WIDTH;
        if (line == a0_line) pred_color = pixel[abs_a0];
        else pred_color = left_reference[line];
        if (pixel[i] != pred_color) { abs_a1 = i; i = 0; }
        else i++;
        if (i >= total_number_of_pixels) i = -1;
    } while (i > 0);
}

if (i == -1) { (a1 is not detected) }
else { (a1 is detected)
    r_a1 = abs_a1 - a0_line*WIDTH;
}
```

**(2) DETECT b1**

```
int i, j, line, a0_line;
i = abs_a0 + 1 - WIDTH;
a0_line = abs_a0/WIDTH;
j = 1;
do {
    line = i/WIDTH;
    if (line == a0_line) pred_color = left_reference[a0_line];
    else pred_color = pixel[abs_a0];
    if (i < 0) {
        if (top_reference[i+WIDTH] != pixel[abs_a0] &&
            top_reference[i+WIDTH-1] != top_reference[i+WIDTH])
            { abs_b1 = i; j = 0; }
    } else {
        if (i%WIDTH == 0 && pixel[i] != pred_color && left_reference[line] != pixel[i])
            { abs_b1 = i; j = 0; }
        else if (pixel[i] != pred_color && pixel[i-1] != pixel[i])
            { abs_b1 = i; j = 0; }
    }
    i++;
} while (j > 0);
```

```

    { abs_b1 = i; j = 0; }
    else i++;
}
if (i > abs_a0) j = -1;
} while (j > 0);

if (j == -1) { (b1 is not detected) }
else { (b1 is detected)
    r_b1 = abs_b1 - (a0_line - 1)*WIDTH;
}

```

where,

pixel[z] : Color of zth pixel from the top left in MB.

left\_reference[z] : Color of zth pixel from the top in Left reference.

top\_reference[z] : Color of zth pixel from the left-end in Top reference.

total\_number\_of\_pixel : Total number of pixels in a MB.

e.g. 256 ( CR = 1 ), 64 ( CR = 1/2 ), 16 ( CR = 1/4 ).

### 3.2.1.5 Binary shape coding with Motion Compensation(MC)

#### 3.2.1.5.1 Framework

Alpha plane is encoded per 16x16 alpha block and syntax is modified to encode the alpha data with the motion texture.

#### 3.2.1.5.2 Description

An alpha plane is encoded per 16x16 alpha block. The alpha coding is strongly coupled to texture coding and done by not only intra frame coding but also inter frame coding with motion compensation. This scheme contribute to both bit reduction and low delay (enable macroblock base operation) compared to that described in VM 2.2.

The alpha block is intra or inter frame coded with motion compensation. The motion vector is the same as YUV macroblock and residual of the alpha block is never coded.

#### Intra / inter alpha coding decision

The motion compensation is applied to binary shape with the same scheme as the luminance macroblock except padding is not performed. The compensated alpha block is clamped before calculating alpha prediction error. The selection of intra or inter alpha coding depends on the alpha prediction error and intra/inter mode of YUV macroblock at the same spatial position of the alpha block.

The alpha block is inter-coded if all the following conditions are satisfied. Otherwise it is intra-coded.

- The 16x16 alpha block is divided into sixteen 4x4 alpha sub-blocks. The alpha prediction error of 4x4 alpha sub-block is the summation of absolute prediction error over the 4x4 alpha sub-block.

Each of the alpha prediction error of 4x4 alpha sub-block is less than or equal to a pre-determined threshold  $TH_{intra}$ .

- The compensated alpha block is not all 0.

- The compensated alpha block is not all 255 and the previous reconstructed block is all 255.
- YUV macroblock is inter coded.

### Inter alpha block coding

The motion compensation is described in previous section. No residual of the alpha block is coded, so only `first_MMR_code` is coded for the alpha block. Note that VLC of `first_MMR_code` of the alpha inter-coded is different from that of the alpha intra-coded.

### Clamping of coded alpha

Alpha values are rounded to the nearest value 0 or 255. It eliminates intermediate values between 0 and 255 that are produced by overlapped motion compensation.

### 3.2.1.6 Size conversion (Rate control)

Binary shape information can be size-converted for rate control and rate reduction. Therefore, if rate control is required, size-conversion process is carried out every MB except All white MB and All black MB.

1. The conversion ratio(CR) is 1 (original size) or 1/2 or 1/4, and CR information is VLC coded (see Table M1).
2. Fig.M6 shows the size-conversion procedure. 16x16 MB is down-sampled to 8x8 or 4x4, and up-sampled to 16x16 again. Down and up sampling filters are used every MB, and they are as follows (Here, "1" means "black", and "0" means "white").

#### [Down sampling]

\* From 16x16 to 8x8 (from "O" to "X" in Fig.M7(a))

If the average of pixel values in a 2x2 pixels block is equal to or larger than 0.5, the pixel value of the down-sampled block is set to "1". Otherwise, the pixel value of the down-sampled block is set to "0".

\* From 16x16 to 4x4

If the average of pixel values in a 4x4 pixels block is equal to or larger than 0.5, the pixel value of the down-sampled block is set to "1". Otherwise, the pixel value of the down-sampled block is set to "0".

#### [Up sampling]

The pixel value of the up-sampled block is calculated by bilinear interpolation (from "O" to "X" in Fig.M7(b)). If a bilinear-interpolated pixel value is equal to or larger than 0.5, the interpolated pixel value is set to "1". Otherwise, the interpolated pixel value is set to "0". In the case of interpolating MB boundary pixel values, the imaginary pixel whose value is equal to that of the nearest down-sampled pixel inside the MB is regarded to exist outside the MB (see Fig.M7(b)).

3. For Top reference and Left reference, the following size-conversion processes (from 16 pixels to 16/N pixels : CR=1/N) are used (C program):

```
int i, j, tmp;
for ( i = 0; i < 16/N; i++ ) {
    tmp = 0;
    for ( j = 0; j < N; j++ ) tmp += top_reference[N*i + j];
```

```

top_reference_N[i] = (int)(( tmp + N/2 ) / N );
tmp = 0;
for ( j = 0; j < N; j++ ) tmp += left_reference[N*i + j];
left_reference_N[i] = (int)(( tmp + N/2 ) / N );
}

```

where,

$\text{left\_reference}[z]$  : Color of  $z$ th pixel from the top in Left reference.  
 $\text{left\_reference\_N}[z]$  : Color of  $z$ th pixel from the top in down-sampled ( $CR=1/N$ ) Left reference.  
 $\text{top\_reference}[z]$  : Color of  $z$ th pixel from the left-end in Top reference.  
 $\text{top\_reference\_N}[z]$  : Color of  $z$ th pixel from the left-end in down-sampled ( $CR=1/N$ ) Top reference.

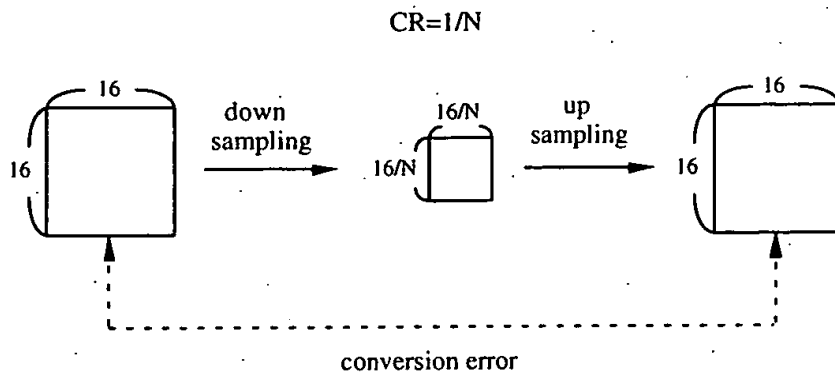
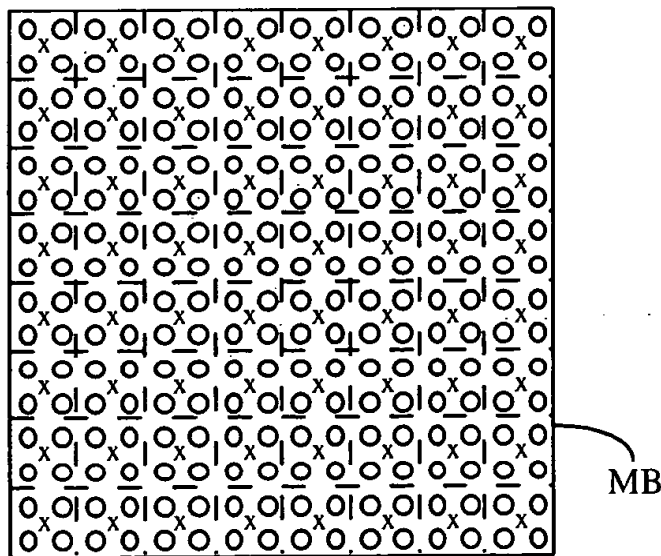
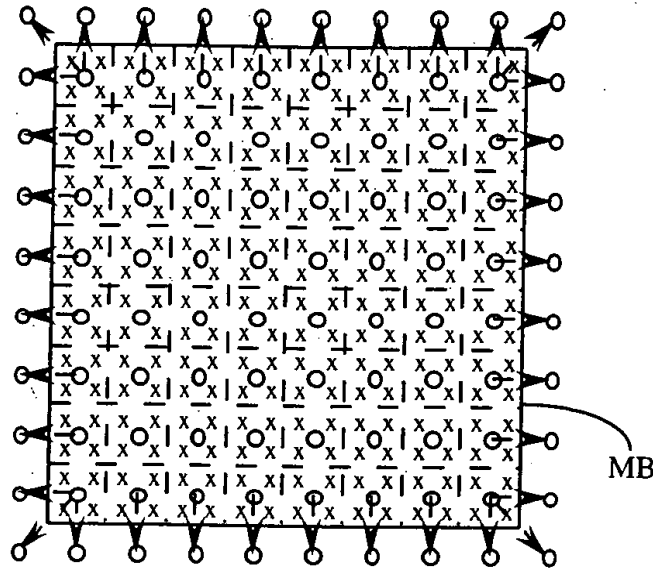


Fig.M6 Size-conversion



(a) Down sampling



(b) Up sampling

Fig. M7 Down sampling / Up sampling disposition

4. Fig.M8 shows the flow chart for determining CR. Error-PB in this chart is regulated as the result of the following three steps.

- (1) Sum of the conversion error for each pixel is calculated every 4x4 pixels block(PB). Here, the conversion error means the absolute difference between the value of a pixel in an original MB and that in a reconverted MB. If the above-mentioned sum is larger than the predetermined threshold TH1, this PB becomes "Error-PB" (in this case, 4x4 block). If there is no 4x4 Error-PB in a MB, the next step is carried out.
- (2) Sum of the conversion error for each pixel is calculated every 8x8 PB. If the sum is larger than the predetermined threshold TH2, this PB becomes "Error-PB" (in this case, 8x8 block). If there is no 8x8 Error-PB in a MB, the next step is carried out.
- (3) Sum of the conversion error for each pixel is calculated every 16x16 PB (in this case, PB is MB itself). If the sum is larger than the predetermined threshold TH3, this PB becomes "Error-PB" (in this case, 16x16 block).

Threshold for each step is changed in accordance with CR as follows.

CR	TH1 (4x4)	TH2 (8x8)	TH3 (16x16)
1/2	$16x(\alpha TH+16)$	$64x(\alpha TH+4)$	$256x\alpha TH$
1/4	$16x(\alpha TH+32)$	$64x(\alpha TH+8)$	$256x\alpha TH$

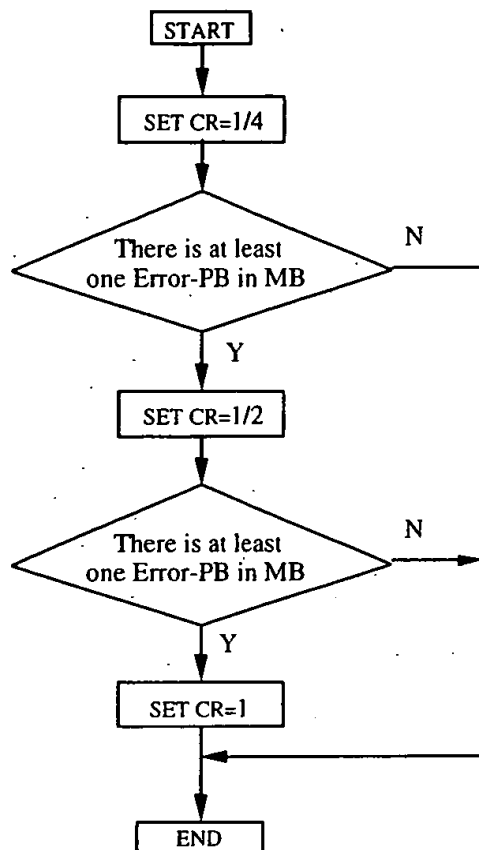


Fig.M8 CR determination algorithm

- Modified MMR is carried out for each size-converted MB whose size is determined by the algorithm shown in Fig.M8.
- If the down-sampled MB becomes All white MB or All black MB, modified MMR is not carried out, and only the code word first\_MMR\_code is transmitted.

### 3.2.1.6 Rounding process for bit-rate control

The following rounding process can be used to control the number of bits. The rounding process starts at a 4x4 alpha sub-block. A 4x4 alpha sub-block is rounded to all '0' or all '255' when the sum of the rounded error is smaller than or equal to a pre-determined threshold ( $TH_{QT}$ ). If the 4x4 alpha sub-block is not rounded, it is further subdivided into four 2x2 alpha sub-blocks. The rounding process is then performed to each of the 2x2 alpha sub-blocks while the sum of the rounded error is below than or equal to  $TH_{QT}$ . The rounding process is given by the following pseudo-code.

```
if ( $\Sigma x \leq TH_{QT}$ )
    x = 0 for all alpha of 4x4 block;
if ( $\Sigma x \geq 16*255-TH_{QT}$ )
    x = 255 for all alpha of 4x4 block;
if (4x4 block is not rounded) {
    int i, dis[4], alpha[4], sum_dis = 0;
    for (i = 0; i < 4; ++i)
        if ( $\Sigma y < 4*255/2$ ) /* y is a alpha value of i-th 2x2 block */
            { dis[i] =  $\Sigma y$ ; alpha[i] = 0; }
        else
            { dis[i] =  $4*255 - \Sigma y$ ; alpha[i] = 255; }
    do {
        int min_dis = 1 << (8*sizeof(int)-2); /* maximum integer value */
        int j;
        for (i = 0; i < 4; ++i) /* search not rounded 2x2 block */
            if ((min_dis > dis[i]) && (dis[i] > 0))
                { min_dis = dis[i]; j = i; }
        if (sum_dis + min_dis <=  $TH_{QT}$ ) {
            sum_dis += min_dis;
            y = alpha[j] for all alpha of j-th 2x2 block;
            min_dis = dis[j] = 0;
        }
    } while (min_dis == 0);
}
```

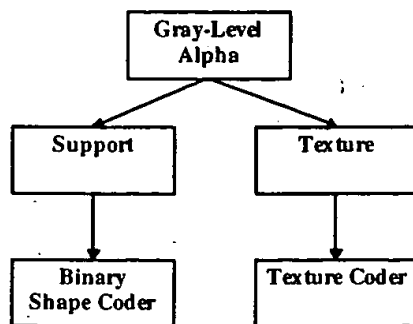
The bit rate of alpha plane is controlled by threshold  $\alpha_{TH}$ . The following parameters are recommended for possible core experiments requiring lossy shape coding.

- $TH_{QT} = 16 * \alpha_{TH}$
- $\alpha_{TH} = 0$  (lossless), 16, 32 and 64  
 $\alpha_{TH}$  should be smaller than 128 to prevent undesirable rounding results.

### 3.2.2 Grey Scale Shape Coding

Gray-level alpha plane is encoded as its support function and the alpha values on the support. The support function is encoded by binary shape coding as described in Section 3.2.1 and the alpha values are encoded as texture with arbitrary shape.





1. The support is obtained by thresholding the gray-level alpha plane by 0.
2. The alpha values are partitioned into 16x16 blocks and encoded the same way as the luminance (see Sections 3.3.4.2 and 3.4). The 16x16 blocks of alpha values are referred to as alpha macroblock hereafter. The encoded data of an alpha macroblock are appended to the end of the corresponding (texture) macroblock, or the encoded macroblock texture in the case of separate motion-texture mode. The formats and syntax of the encoded alpha macroblocks are described in the following.

- I-VOP and P-VOP

CODA	CBPA	Alpha Block Data
------	------	------------------

For I-VOP, CODA is 1 if the alpha values in the alpha macroblock are all 255 and 0 otherwise. For P-VOP,

```

if alpha_mb_all_opaque {
    if colocated_alpha_mb_all_opaque
        CODA = 1
    else
        CODA = 01
}
else {
    if (MV == 0 && alpha_residue_all_zero)
        CODA = 1
    else
        CODA = 00
}
  
```

When CODA is 1 or 01, no more data are included in the bitstream. CBPA (Coded block pattern for alpha) is the same as CBBY (See Appendix A.1.4). Note that for both I-VOP and P-VOP, the third column of Table B.5 is used. Alpha block data format are the same as block data (see Appendix A.2). Note the alpha macroblocks and blocks with all zero alpha values are not included in the bitstream. The CBPA bit for all zero alpha blocks is set to zero. All of the rest parameters needed for encoding and decoding of alpha macroblocks are the same as in the texture macroblocks.

- B-VOP

CODA	MODBA	CBPBA	Alpha Block Data
------	-------	-------	------------------

CODA is 1 if the alpha values in the alpha macroblock are all 255 and thus no more data are sent to the bitstream. CODA is 0 otherwise. MODBA has the same meaning as MODB (See Appendix B.1.9) and CBPBA is similarly defined as CBPB (see B.1.11) except that it only has four bits.

### 3.3 Motion Estimation and Compensation

In order to perform motion prediction on a per VOP basis, the motion estimation of the blocks on the VOP borders has to be modified from block matching to polygon matching. Furthermore, a special padding technique, i.e., the repetitive padding, is required for the reference VOP. The details of these techniques are described in the following sections.

Since the VOPs have arbitrary shapes rather rectangular shapes, and the shapes change from time to time, some conventions is necessary to ensure the consistency of the motion compensation in the VM.

The absolute (frame) coordinate system is used for referencing all of the VOPs. At each particular time instance, a bounding rectangle that includes the shape of that VOP, as described in section 3.1, is defined. The left and top corner, in their absolute coordinates, of the bounding box is encoded in the VOP spatial reference.

Thus, the motion vector for a particular feature inside a VOP, e.g. a macroblock, refers to the displacement of the feature in absolute coordinates. No alignment of VOP bounding boxes at different time instances is performed.

In addition to the motion estimation and compensation mode, two additional modes are supported, namely, unrestricted and advanced modes are supported. In all three modes, the motion vector search range is up to  $[-64, 63.5]$ . This mode differs from the unrestricted motion mainly by restricting the motion vectors inside the bounding box of the VOP. The advanced mode allows multiple motion vectors in one macroblock and overlapped motion compensation. Note that in all three modes, padding of the VOP up to a rectangle is needed for both motion estimation and compensation.

### **3.3.1 Image Padding Technique**

An image padding technique, repetitive padding, is applied on the reference VOP for performing motion estimation/compensation and on the texture coding for DCT. In this section, the procedure of repetitive padding is described. The details of how the repetitive padding is applied to unrestricted motion estimation/compensation and residual errors are described in each corresponding section.

Repetitive padding process consists of five steps. The reconstructed alpha plane is used for repetitive padding.

- (1) Consider each undefined pixel outside the object boundary a zero pixel.
- (2) Scan each horizontal line of the original image region. Each scan line is possibly composed of two kinds of line segments: zero segments that have all zero pixels within each segment and non-zero segments that have all non-zero pixels within each segment. If there are no non-zero segments, do nothing. Otherwise, there are two situations for a particular zero segment: it can be positioned either between an end point of the scan line and the end point of a non-zero segment, or, between the end points of two different non-zero segments. In the first case, fill all of the pixels in the zero segments with the pixel value of the end point of the non-zero segment. In the second case, fill all of the pixels in the zero segments with the averaged pixel value of the two end points.
- (3) Scan each vertical line of the original image and perform the identical procedure as described in (1) to each vertical scan line.
- (4) If a zero pixel can be filled in by both (2) and (3), the final value takes the average of the two possible values.

- (5) Consider the rest of zero pixels. For any one of them, scan horizontally to find the closest non-zero pixel on the same horizontal scan (if there is a tie, the non-zero pixel to the left of the current pixel is selected), and scan vertically to find the closest non-zero pixels on the same vertical scan (if there is a tie, the non-zero pixel on the top of the current pixel is selected). Replace the zero pixel by the average of these two horizontally and vertically closest non-zero pixels.

The Figure 3.3.1 illustrates the outcome of each of the steps described above.

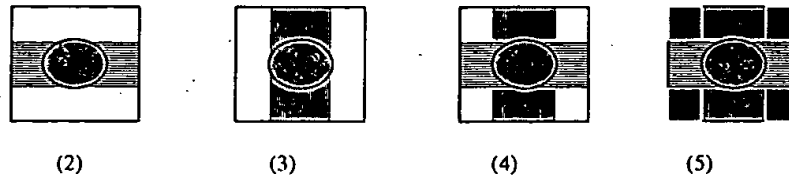


Figure 3.3.1 Illustration of some steps of repetitive padding.

### 3.3.2 Basic Motion Techniques

#### 3.3.2.1 Modified Block (Polygon) Matching

The bounding rectangle of the VOP is first extended on the right-bottom side to multiples of macroblock size. So the size of the bounding rectangle of the luminance VOP is multiples of  $16 \times 16$ , while the size of the chrominance plane is multiples of  $8 \times 8$ . The alpha value of the extended pixels is set to be zero. The macroblocks are formed by dividing the extended bounding rectangles into  $16 \times 16$  blocks. Zero stuffing is used for these extended pixels. SAD (Sum of Absolute Difference) is used as error measure. The original alpha plane for the VOP is used to exclude the pixels of the macroblock that are outside the VOP. SAD is computed only for the pixels with nonzero alpha value. This forms a polygon for the macro block that includes the VOP boundary. Figure 3.3.2 illustrates an example.

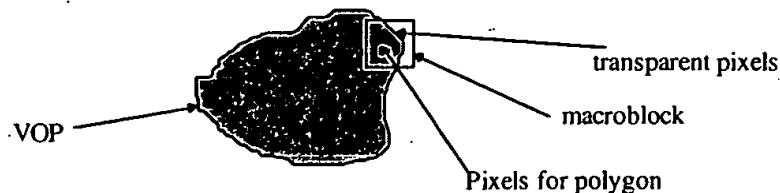


Figure 3.3.2 Polygon matching for an arbitrary shape VOP.

The reference VOP needs to be padded strictly based on its own shape information. For example, when the reference VOP is smaller than the current VOP, the reference is not padded up to the size of the current VOP.

In case the 8x8 advanced prediction mode is chosen and all of the pixels in an 8x8 block are transparent (completely outside the VOP), the SAD is set to be zero and the motion vector for this 8x8 block is zero. No matching needs to be done for this 8x8 block.

### 3.3.2.2 Integer pixel motion estimation

Both 8x8 and 16x16 vectors are obtained from the search algorithm. Only a small amount of additional computation is needed to obtain the 8x8 integer vectors in addition to the 16x16 vectors.

The search is made with integer pixel displacement and for the Y component. The comparisons are made between the incoming block and the displaced block in the previous original VOP. A full search around the original macroblock position is used with a maximum search area depending on the range provided by the f\_code.

$$SAD_N(x, y) = \sum_{i=1, j=1}^{N, N} |original - previous| * (! (Alpha_{original} == 0)),$$

x, y = "up to [-64, 63]", N = 16 or 8

For the zero vector  $SAD_{16}(0,0)$  is reduced to favor the zero vector when there is no significant difference.

$$SAD_{16}(0,0) = SAD_{16}(0,0) - (N_B / 2 + 1)$$

where  $N_B$  = number of macroblock pixels inside VOP. The (x,y) pair resulting in the lowest  $SAD_{16}$  is chosen as the 16x16 integer pixel motion vector, V0. The corresponding SAD is  $SAD_{16}(x,y)$ .

Likewise, the (x,y) pairs resulting in the lowest  $SAD_8(x,y)$  are chosen to give the 4 8x8 vectors V1, V2, V3 and V4.

The 8x8 based SAD for the macroblock is

$$SAD_{K \times 8} = \sum_1^K SAD_8(x, y)$$

where  $0 < K \leq 4$  is the number of 8x8 blocks that do not lie outside of the VOP shape. The following rule, and

$$SAD_{inter} = \min(SAD_{16}(x, y), SAD_{K \times 8})$$

Instead of full search, the 8x8 search is centered around 16x16 vector for the following reasons :

- (i) it is faster,
- (ii) it generally gives better results because of a better OBMC filtering effect and less bits spent for vectors,
- (iii) if the Extended MV Range is used, the search range around the motion vector predictor will be less limited.

### 3.3.2.3 INTRA/INTER mode decision

After the integer pixel motion estimation the coder makes a decision on whether to use INTRA or INTER prediction in the coding. The following parameters are calculated to make the INTRA/INTER decision:

$$MB\_mean = (\sum_{i=1, j=1}^{N_B} original) / N_B$$

$$A = \sum_{i=1, j=1}^{16,16} |original - MB\_mean| * (!(Alpha_{original} == 0))$$

INTRA mode is chosen if:  $A < (SAD_{inter} - 2 * N_B)$

Notice that if  $SAD_{16}(0,0)$  is used, this is the value that is already reduced as explained above.

If INTRA mode is chosen, no further operations are necessary for the motion search. If INTER mode is chosen the motion search continues with half sample search around the V0 position.

### 3.3.2.4 Half sample search

Half sample search is performed for 16x16 vectors as well as for 8x8 vectors. The half sample search is done using the previous reconstructed VOP. The search is performed on the luminance component of the macroblock, and the search area is  $\pm 1$  half sample around the target matrix pointed to by V0, V1, V2, V3 or V4. For the 16x16 search the zero vector sad,  $SAD(0,0)$ , is reduced by  $NB/2+1$  as for the interger search.

The half sample values are found using the interpolation described in Figure 3.3.3 and which corresponds to bilinear interpolation.

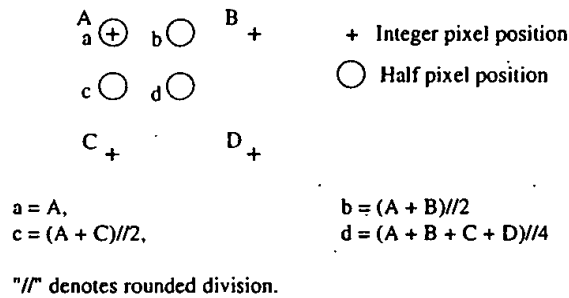


Figure 3.3.3 Interpolation scheme for half sample search.

The vector resulting in the best match during the half sample search is named MV. MV consists of horizontal and vertical components (MVx, MVy), both measured in half sample units.

### 3.3.2.5 Decision on 16x16 or 8x8 prediction mode

SAD for the best half sample 16x16 vector (including subtraction of  $NB/2+1$  if the vector is (0,0)):

$$SAD_{16}(x, y)$$

SAD for the whole macro block for the best half sample 8x8 vectors:

$$SAD_{K \times 8} = \sum_1^K SAD_8(x, y)$$

where  $0 < K \leq 4$  is the number of  $8 \times 8$  blocks that do not lie outside of the VOP shape. The following rule applies:

If:  $SAD_{K \times 8} < SAD_{16} - (N_B / 2 + 1)$ , choose  $8 \times 8$  prediction  
otherwise: choose  $16 \times 16$  prediction

### 3.3.2.6 The motion vector range

- *To be reedited.*

### 3.3.2.7 Differential coding of motion vectors

When using INTER mode coding, the motion vector must be transmitted. The motion vector components (horizontal and vertical) are coded differentially by using a spatial neighborhood of three motion vectors already transmitted (Figure 3.3.4). These three motion vectors are candidate predictors for the differential coding.

In the special cases at the borders of the current VOP the following decision rules are applied:

1. If the macroblock of one and only one candidate predictor is outside of the VOP, it is set to zero.
2. If the macroblocks of two and only two candidate predictors are outside of the VOP, they are set to the third candidate predictor.
3. If the macroblocks of all three candidate predictors are outside of the VOP, they are set to zero.

The motion vector coding is performed separately on the horizontal and vertical components.

For each component, the median value of the three candidates for the same component is computed:

$$Px = \text{Median}(MV1x, MV2x, MV3x)$$

$$Py = \text{Median}(MV1y, MV2y, MV3y)$$

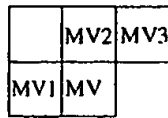
For instance, if  $MV1=(-2,3)$ ,  $MV2=(1,5)$  and  $MV3=(-1,7)$ , then  $Px = -1$  and  $Py = 5$ .

The Variable Length Codes for the vector differences  $MVDx$  and  $MVDy$  are listed in Table B.7.

$$MVDx = MVx - Px$$

$$MVDy = MVy - Py$$

For prediction of  $8 \times 8$  vectors see section 3.3.4.



MV : Current motion vector  
 MV1: Previous motion vector  
 MV2: Above motion vector  
 MV3: Above right motion vector

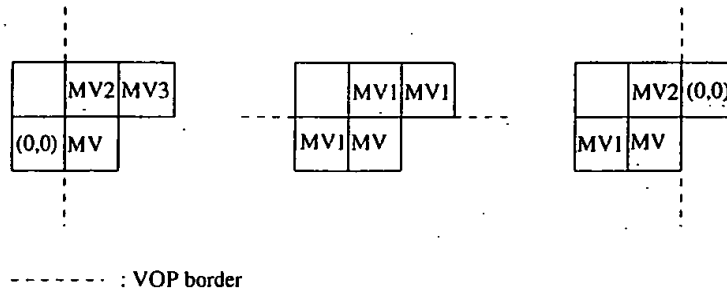
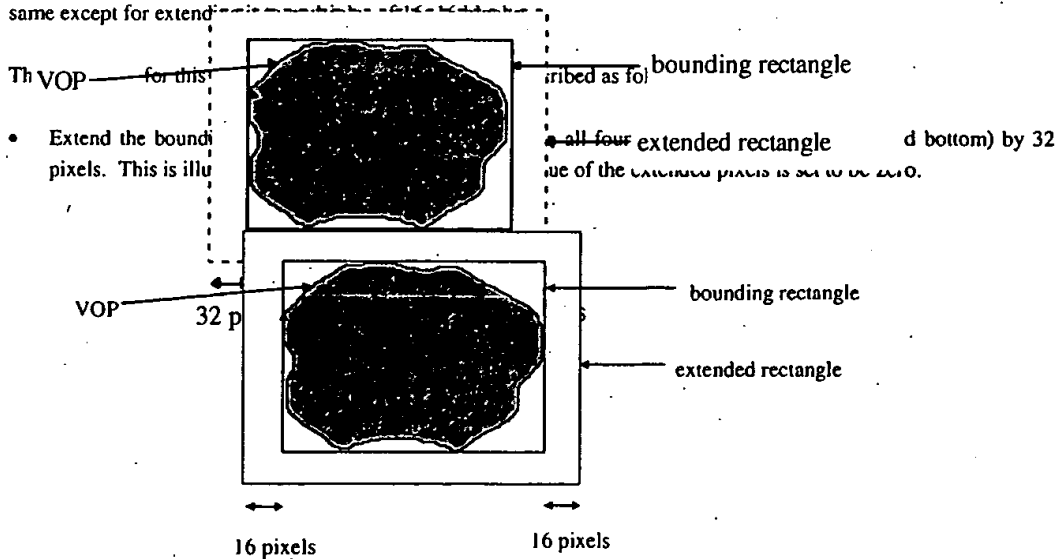


Figure 3.3.4 Motion vector prediction.

### 3.3.3 Unrestricted Motion Estimation/Compensation

#### 3.3.3.1 Motion vectors over VOP boundaries

An unrestricted motion estimation mode is used for VOP motion estimation and compensation. The technique is to improve the motion estimation techniques, especially for VOP-based coding schemes. In this technique, the error signal is generated by extending the reference VOP to enough size, padding the extended VOP, applying motion estimation and compensation, and taking the difference of the original and the estimated signals. Note that padding is performed only on the reference VOP. Target VOP remains the same except for extended



Repetitive padding on all transparent (alpha == 0) pixels.

Figure 3.3.5 Illustration of extending reference VOP.

- Pad the extended regions using repetitive padding. Use the padded VOP as the new reference VOP.
- Apply modified block (polygon) matching described in Section 3.3.2.1 to compute the motion vectors.

### 3.3.4 Advanced prediction mode

#### 3.3.4.1 Formation of the motion vectors

One/four vectors decision is indicated by the MCBPC codeword for each macroblock. If only one motion vector is transmitted for a certain macroblock, this is defined as four vectors with the same value. If MCBPC indicates that four motion vectors are transmitted for the current macroblock, the information for the first motion vector is transmitted as the codeword MVD and the information for the three additional motion vectors is transmitted as the codewords MVD<sub>2-4</sub>.

The vectors are obtained by adding predictors to the vector differences indicated by MVD and MVD<sub>2-4</sub> in a similar way as when only one motion vector per macroblock is present, according to the decision rules given in section 3.3. Again the predictors are calculated separately for the horizontal and vertical components. However, the candidate predictors MV1, MV2 and MV3 are redefined as indicated in Figure 3.3.6. If only one vector per macroblock is present, MV1, MV2 and MV3 are defined as for the 8\*8 block numbered 1 in Figure B.3.

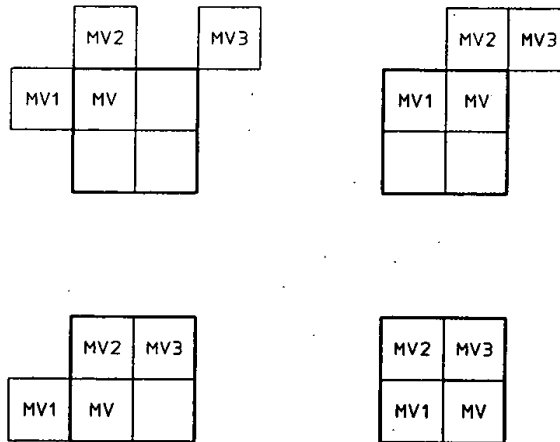


Figure 3.3.6 Redefinition of the candidate predictors MV1, MV2 and MV3 for each of the luminance blocks in a macroblock

If four vectors are used, each of the motion vectors is used for all pixels in one of the four luminance blocks in the macroblock. The numbering of the motion vectors is equivalent to the numbering of the four luminance blocks as given in Figure B.3. Motion vector MVD<sub>CHR</sub> for both chrominance blocks is derived



by calculating the sum of the  $K$  luminance vectors, that corresponds to  $K$  8x8 blocks that do not lie outside the VOP shape and dividing this sum by  $2^*$ ; the component values of the resulting sixteenth/twelfth/eighth/fourth sample resolution vectors are modified towards the nearest half sample position as indicated in Table 3.3.1.a/b/c/d.

**Table 3.3.1.a**

Modification of sixteenth sample resolution chrominance vector components

sixteenth pixel position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	/16
resulting position	0	0	0	1	1	1	1	1	1	1	1	1	1	1	2	2	/2

**Table 3.3.1.b**

Modification of twelfth sample resolution chrominance vector components

twelfth pixel position	0	1	2	3	4	5	6	7	8	9	10	11	/12
resulting position	0	0	1	1	1	1	1	1	1	1	2	2	/2

**Table 3.3.1.c**

Modification of eighth sample resolution chrominance vector components

eighth pixel position	0	1	2	3	4	5	6	7	/8
resulting position	0	0	1	1	1	1	1	2	/2

**Table 3.3.1.d**

Modification of fourth sample resolution chrominance vector components

fourth pixel position	0	1	2	3	/4
resulting position	0	1	1	2	/2

Half sample values are found using bilinear interpolation as described in section 6.1.2. The prediction for luminance is obtained by overlapped motion compensation as described above. The prediction for chrominance is obtained by applying the motion vector  $MVD_{CHR}$  to all pixels in the two chrominance blocks (as it is done in the default prediction mode).

The predictor for MVD and  $MVD_{2,4}$  is defined as the median value of the vector components  $MV1$ ,  $MV2$  and  $MV3$  as defined in section 3.3.2.7.

### 3.3.4.2 Overlapped motion compensation for luminance

Each pixel in an 8\*8 luminance prediction block is a weighted sum of three prediction values, divided by 8 (with rounding). In order to obtain the three prediction values, three motion vectors are used: the motion vector of the current luminance block, and two out of four "remote" vectors:

- the motion vector of the block at the left or right side of the current luminance block;
- the motion vector of the block above or below the current luminance block.

For each pixel, the remote motion vectors of the blocks at the two nearest block borders are used. This means that for the upper half of the block the motion vector corresponding to the block above the current block is used, while for the lower half of the block the motion vector corresponding to the block below the current block is used (see Figure 3.3.8). Similarly, for the left half of the block the motion vector corresponding to the block at the left side of the current block is used, while for the right half of the block the motion vector corresponding to the block at the right side of the current block is used (see Figure 3.3.9).

The creation of each pixel,  $\bar{p}(i, j)$ , in an 8\*8 luminance prediction block is governed by the following equation:

$$\bar{p}(i, j) = (q(i, j) \times H_0(i, j) + r(i, j) \times H_1(i, j) + s(i, j) \times H_2(i, j) + 4) / 8,$$

where  $q(i, j)$ ,  $r(i, j)$ , and  $s(i, j)$  are the pixels from the referenced picture as defined by

$$q(i, j) = p(i + MV_x^0, j + MV_y^0),$$

$$r(i, j) = p(i + MV_x^1, j + MV_y^1),$$

$$s(i, j) = p(i + MV_x^2, j + MV_y^2).$$

Here,  $(MV_x^0, MV_y^0)$  denotes the motion vector for the current block,  $(MV_x^1, MV_y^1)$  denotes the motion vector of the block either above or below, and  $(MV_x^2, MV_y^2)$  denotes the motion vector either to the left or right of the current block as defined above.

The matrices  $H_0(i, j)$ ,  $H_1(i, j)$  and  $H_2(i, j)$  are defined in Figure 3.3.7, Figure 3.3.8, and Figure 3.3.9, where  $(i, j)$  denotes the column and row, respectively, of the matrix.

If one of the surrounding blocks was not coded, the corresponding remote motion vector is set to zero. If one of the surrounding blocks was coded in INTRA mode, the corresponding remote motion vector is replaced by the motion vector for the current block. If the current block is at the border of the VOP and therefore a surrounding block is not present, the corresponding remote motion vector is replaced by the current motion vector. In addition, if the current block is at the bottom of the macroblock, the remote motion vector corresponding with an 8\*8 luminance block in the macroblock below the current macroblock is replaced by the motion vector for the current block.

The weighting values for the prediction are given in Figure 3.3.7, Figure 3.3.8, and Figure 3.3.9.

4	5	5	5	5	5	5	4
5	5	5	5	5	5	5	5
5	5	6	6	6	6	5	5
5	5	6	6	6	6	5	5
5	5	6	6	6	6	5	5
5	5	6	6	6	6	5	5
5	5	5	5	5	5	5	5
4	5	5	5	5	5	5	4

Figure 3.3.7 Weighting values,  $H_0$ , for prediction with motion vector of current luminance block

2	2	2	2	2	2	2	2
1	1	2	2	2	2	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2

Figure 3.3.8 Weighting values,  $H_1$ , for prediction with motion vectors of the luminance blocks on top or bottom of current luminance block

2	1	1	1	1	1	1	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	1	1	1	1	1	1	2

Figure 3.3.9 Weighting values,  $H_2$ , for prediction with motion vectors of the luminance blocks to the left or right of current luminance block

### 3.4 Texture Coding

The intra VOPs and the residual data after motion compensation is coded using the same 8x8 block DCT scheme. DCT is done separately for each of the luminance and chrominance planes. When shape of the VOP is arbitrary, the macroblocks that belong to the arbitrary shape of the VOP are treated as described below. There are two types of macroblocks that belong to an arbitrarily shaped VOP: 1) those that lie completely inside the VOP shape and 2) those that lie on the boundary of the shape. The macroblocks that lie completely inside the VOP are coded using a technique identical to the technique used in H263. The intra 8x8 blocks that belong to the macroblocks lying on the border of the VOP shape are first padded as described in the motion estimation/compensation section. For padding of chroma blocks, a 16x16 alpha block is decimated by discarding every other row and column of pixels, starting from the second row and second column. For residue blocks, the region outside the VOP within the blocks are padded with zero. Padding is performed separately for each of the luminance and chrominance 8x8 blocks by using the reconstructed alpha values of the luminance or chrominance in this 8x8 block. If all the pixels in an 8x8 block are transparent, their values are replaced by zero. These blocks are then coded in a manner identical to the interior blocks. The macroblocks that do not belong to the arbitrary shape but inside the bounding box of a VOP are not coded at all.

### 3.4.1 DCT

A separable 2-dimensional Discrete Cosine Transform (DCT) is used.

### 3.4.2 H.263 Quantization Method

The quantization parameter  $QP$  may take integer values from 1 to 31. The quantization stepsize is  $2 \times QP$ .

$COF$  A transform coefficient to be quantized.  
 $LEVEL$  Absolute value of the quantized version of the transform coefficient.  
 $COF'$  Reconstructed transform coefficient.

**Quantization:**

For INTRA:  $LEVEL = |COF| / (2 \times QP)$   
For INTER:  $LEVEL = (|COF| - QP / 2) / (2 \times QP)$

**Dequantization:**

$|COF'| = 0$ , if  $LEVEL = 0$   
 $|COF'| = 2 \times QP \times LEVEL + QP$ , if  $LEVEL \neq 0$ ,  $QP$  is odd  
 $|COF'| = 2 \times QP \times LEVEL + QP - 1$ , if  $LEVEL \neq 0$ ,  $QP$  is even

The sign of  $COF$  is then added to obtain  $COF'$ :  $COF' = \text{Sign}(COF) \times |COF'|$

The DC coefficient of an INTRA block is quantized as described below. 8 bits are used for the quantized DC coefficient.

**Quantization:**

$LEVEL = COF // 8$

**Dequantization:**

$COF' = LEVEL \times 8$

### 3.4.3 MPEG Quantization Method

#### 3.4.3.1 Quantization of Intra Macroblocks

##### DC Coefficient

The quantizer step-size for the DC coefficient of the luminance and chrominance components is 8. Thus, the quantized DC value, QDC, is calculated as:

$$QDC = dc // 8$$

where "dc" is the 11-bit unquantized value from the DCT.

### AC Coefficients

AC coefficients  $ac[i][j]$  are first quantised by individual quantization factors,

$$ac_{-}[i][j] = (16 * ac[i][j]) // w_1[i][j]$$

where  $w_1[i][j]$  is the  $[i][j]$ th element of the default Intra quantizer matrix, which for this VM carries a value of 16 except for DC coefficient location which carries a value 8; this is referred to as flat matrix. The resulting  $ac_{-}[i][j]$  is limited to the range  $[-2048, 2047]$ .

An example of non-flat intra quantization matrix which can be alternatively used as default is provided for guidance in Figure 3.4.3.1 and if used should be clearly stated in core experiment comparisons against VM.

8	17	18	19	21	23	25	27
17	18	19	21	23	25	27	28
20	21	22	23	24	26	28	30
21	22	23	24	26	28	30	32
22	23	24	26	28	30	32	35
23	24	26	28	30	32	35	38
25	26	28	30	32	35	38	41
27	28	30	32	35	38	41	45

Figure 3.4.3.1 - Example Intra quantizer matrix

The step-size for quantizing the scaled DCT coefficients,  $ac_{-}[i][j]$ , is derived from the quantization parameter, QP.

The quantized level  $QAC[i][j]$  is given by:

$$QAC[i][j] = (ac_{-}[i][j] + \text{sign}(ac_{-}[i][j]) * ((p * QP) // q)) / (2 * QP)$$

where,  $QAC[i][j]$  is limited to the range  $[-127..127]$ .

For this VM  $p=3$ , and  $q=4$ .

### 3.4.3.2 Quantization of Non Intra Macroblocks

#### Forward Quantization:

Non-intra macroblocks in P- and B- VOPs are quantized with a uniform quantizer that has a dead-zone about zero. The default quantization matrix carries a value of 16 for each entry and is referred to as flat matrix.

An example of non-flat nonintra quantization matrix which can be alternatively used as default is provided for guidance in Figure and if used should be clearly stated in core experiment comparisons against VM.

16	17	18	19	20	21	22	23
17	18	19	20	21	22	23	24
18	19	20	21	22	23	24	25
19	20	21	22	23	24	26	27
20	21	22	23	25	26	27	28
21	22	23	24	26	27	28	30
22	23	24	26	27	28	30	31
23	24	25	27	28	30	31	33

Figure 3.4.3.2 - Non-intra quantizer matrix

The step-size for quantizing both the scaled DC and AC coefficients is derived from the quantization parameter, QP.

$$ac-[i][j] = (16 * ac[i][j]) // w_N[i][j]$$

where:

$w_N[i][j]$  is the non-intra quantizer matrix

$$QAC[i][j] = ac-[i][j] / (2 * QP)$$

QAC [i][j] is limited to the range [-128..127].

### 3.4.3.3 Inverse Quantization of Intra and Non Intra Macroblocks

QF[i][j] are levels decoded from the bitstream for a block. The following equations describe the process of inverse quantization. Two main cases are identified, one when macroblock being decoded is intra and the other when it is nonintra.

```
for (v=0; v<8;v++) {
    for (u=0; u<8;u++) {
        if ( (u==0) && (v==0) && (MB intra) ) {
            F'[v][u] = 8 * QF[0][0];
        } else {
            if ( MB intra ) {
                F'[v][u] = ( QF[v][u] * w_l[v][u] * QP * 2 ) / 16;
            } else {
                F'[v][u] = ( ( ( QF[v][u] * 2 ) + Sign(QF[v][u]) ) * w_N[v][u]
                    * QP ) / 16;
            }
        }
    }
}
```

### 3.4.3.4 DC Prediction of DC coefficients in Intra Macroblocks

After the DC coefficient of a block has been quantized to 8 bits, it is coded lossless by a DPCM technique. Coding of the luminance blocks within a macroblock follows the normal scan of Figure B.3. Thus the DC value of block 4 becomes the DC predictor for block 1 of the following macroblock, resulting in the zig-zag scan order shown in Figure 3.4.3.3. The following details the process for DC prediction of DC coefficients in intra macroblocks:

1. Separate predictors are used for Y, U and V.
2. Predictors are initialised to 128 before the first macroblock at the left edge of the VOP macroblock row.
3. For U/V, the predictor for the current macroblock is given by the U/V DC value in the macroblock to the left. This predictor is set to 128 if the macroblock to the left is a P macroblock or is fully outside the VOP boundary.
4. For each Y macroblock, the DC predictor is given by the "last used" DC value in the macroblock to the left. This predictor is set to 128 if the macroblock to the left is a P macroblock or contains no Y blocks at all (fully-outside VOP boundary). The "last used" DC value is defined to be the DC value of the block with highest block number which is not fully outside the VOP.

5. For Y, intra DC prediction proceeds in block order: 1,2,3,4 (a zig-zag). The first block which is not fully outside the VOP is predicted using the predictor from step 4. Each subsequent block that is present is predicted from the previous block which is present. Note that DC predictors are not reset to 128 within a macroblock because any other block will generally provide a better predictor than 128.

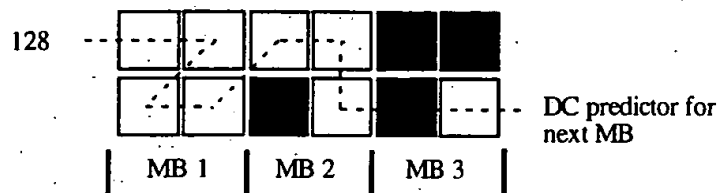


Figure 3.4.3.3. Zigzag scanning of macroblocks for DC prediction.

At the decoder, the original quantized DC values are exactly recovered by following the inverse procedure.

The differential DC values thus generated are categorised according to their "size" as shown in the Tables T1 and T2

Table T1 --- Variable length codes for DC size luminance

vlc code	DC size Luminance
100	0
00	1
01	2
101	3
110	4
1110	5
1111 0	6
1111 10	7
1111 110	8

**Table T2 -- Variable length codes for DC size chrominance**

Vlc code	DC size chrominance
00	0
01	1
10	2
110	3
1110	4
1111 0	5
1111 10	6
1111 110	7
1111 1110	8

For each category additional bits are appended to the SIZE code to uniquely identify which difference in that category actually occurred (table 8.3). The additional bits thus define the signed amplitude of the difference data. The number of additional bits (sign included) is equal to the SIZE value.

**Table T3. Differential DC additional codes**

DIFFERENTIAL DC	SIZE	ADDITIONAL CODE
-255 to -128	8	00000000 to 01111111
-127 to -64	7	0000000 to 0111111
-63 to -32	6	000000 to 011111
-31 to -16	5	00000 to 01111
-15 to -8	4	0000 to 0111
-7 to -4	3	000 to 011
3 to -2	2	00 to 01
-1	1	0
0	0	
1	1	1
2 to 3	2	10 to 11
4 to 7	3	100 to 111
8 to 15	4	1000 to 1111
16 to 31	5	10000 to 11111
32 to 63	6	100000 to 111111
64 to 127	7	1000000 to 1111111
128 to 255	8	10000000 to 11111111

### 3.4.4 VLC encoding of quantized transform coefficients

The 8x8 blocks of transform coefficients are scanned with "zigzag" scanning as listed in Figure 3.4.4.1.

```

1  2  6  7 15 16 28 29
3  5  8 14 17 27 30 43
4  9 13 18 26 31 42 44
10 12 19 25 32 41 45 54
11 20 24 33 40 46 53 55

```



21	23	34	39	47	52	56	61
22	35	38	48	51	57	60	62
36	37	49	50	58	59	63	64

**Figure 3.4.4.1. Zigzag scanning pattern.**

A three dimensional variable length code is used to code transform coefficients. An EVENT is a combination of three parameters:

LAST	0: There are more nonzero coefficients in the block. 1: This is the last nonzero coefficient in the block.
RUN	Number of zero coefficients preceding the current nonzero coefficient.
LEVEL	Magnitude of the coefficient.

The most commonly occurring combinations of (LAST, RUN, LEVEL) are coded with variable length codes given in Appendix B. The remaining combinations of (LAST, RUN, LEVEL) are coded with a 22 bit word consisting of :

ESCAPE	7 bit	
LAST	1 bit	(0: Not last coefficient, 1: Last nonzero coefficient)
RUN	6 bit	
LEVEL	8 bit	

The code words for these fixed length ESCAPE codes are described in Appendix B.

### **3.5. Prediction and Coding of B-VOPs**

Macroblocks in B-VOPs can be coded either using H.263 like B-block coding or by MPEG-1 like B-picture macroblock coding. The main difference is in the amount of motion vector and quantization related overhead needed. The MBTYPE with H.263 like B-block coding is referred to as direct prediction, besides which, the forward, the backward and the interpolated prediction modes of MPEG-1 B-pictures are supported. The syntax and semantics for macroblock and block layer for B-VOPs are presented in Appendix B. The encoding issues for B-VOPs are discussed next.

#### **3.5.1. Direct Coding**

This coding mode uses direct bidirectional motion compensation derived by extending H.263 approach of employing P-picture macroblock motion vectors and scaling them to derive forward and backward motion vectors for macroblocks in B-picture. This is the only mode which makes it possible to use motion vectors on 8x8 blocks, of course, this is only possible when the co-located macroblock in the following P-VOP uses 8x8 MV mode. As per H.263, using B-frame syntax, only one delta motion vector is allowed per macroblock. Figure 3.5.1 shows scaling of motion vectors.

The first extension of the H.263 approach is that bidirectional predictions can be made for a full block/macroblock as in MPEG-1. The second extension of H.263 is that instead of allowing interpolation of only one intervening VOP, more than one VOPs can be interpolated. Of course, if the prediction is poor due to fast motion or large interframe distance, other motion compensation modes can be chosen.

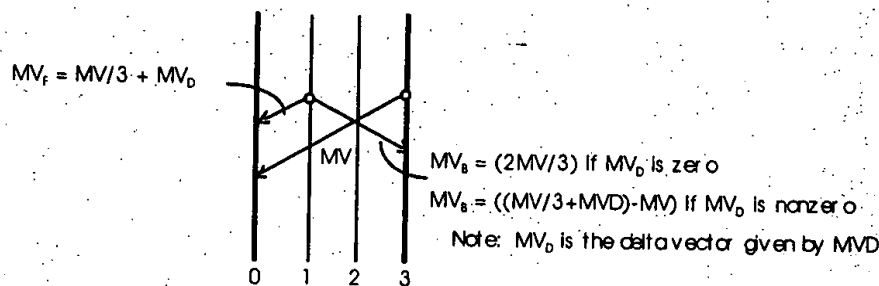


Figure 3.5.1 Direct Bidirectional Prediction

#### 3.5.1.1. Calculation of vectors

The calculation of forward and backward motion vectors involves linear scaling of the collocated block in temporally next P-VOP, followed by correction by a delta vector, and is thus practically identical to the procedure followed in H.263. The only slight change is that here we are dealing with VOPs instead of pictures, and instead of only a single B-picture between a pair of reference pictures, multiple B-VOPs are allowed between a pair of reference VOPs. As in H.263, the temporal reference of the B-VOP relative to difference in the temporal reference of the pair of reference VOPs is used to determine scale factors for computing motion vectors which are corrected by the delta vector.

The forward and the backward motion vectors are  $MV_f$  and  $MV_b$  and are given in half sample units as follows.

$$MV_f = (TR_b \times MV) / TR_D + MV_D$$

$$MV_b = ((TR_b - TR_D) \times MV) / TR_D \quad \text{if } MV_D \text{ is equal to } 0$$

$$MV_b = MV_f - MV \quad \text{if } MV_D \text{ is not equal to } 0$$

Where  $MV$  is the direct motion vector of a macroblock in P-VOP with respect to a reference VOP,  $TR_b$  is the difference in temporal reference of the B-VOP and the previous reference VOP.  $TR_D$  is the difference in temporal reference of the temporally next reference VOP with temporally previous reference VOP, assuming B-VOPs or skipped VOPs in between.

#### 3.5.1.2. Generating Prediction Block

The process of generating a prediction block is fairly trivial and simply consists of using computed forward and backward motion vectors to obtain appropriate blocks from reference VOPs and averaging these blocks. Irrespective of whether the direct prediction motion vectors are derived by scaling of a single motion vector or four 8x8 motion vectors per block, motion compensation is performed individually on 8x8 blocks to generate a macroblock. In case for a macroblock only a single motion vector was available to compute direct prediction motion vector, it is simply repeated for each of the 8x8 blocks forming the macroblock. The main difference with H.263 is that there are no constraints in the amount of region within a block that can be bidirectionally predicted; each entire macroblock can be bidirectionally predicted.

The direct coding mode does not allow quantizer change and thus the quantizer value for previous coded macroblock is used.

#### 3.5.2. Forward Coding

Forward coding mode uses forward motion compensation in the same manner as in MPEG-1/2 with the difference that a VOP is used for prediction instead of a picture. Only one motion vector in half sample

units is employed for a 16x16 macroblock being coded. Chrominance vectors are derived by scaling of luminance vectors as in MPEG-1/2.

This coding mode also allows switching of quantizer from the one previously in use. Specification of DQUANT, a differential quantizer involves a 2-bit overhead as discussed earlier.

### 3.5.3. Backward Coding

Backward coding mode uses backward motion compensation in the same manner as in MPEG-1/2 with the difference that a VOP is used for prediction instead of a picture. Only one motion vector in half sample units is employed for a 16x16 macroblock being coded. Chrominance vectors are derived by scaling of luminance vectors as in MPEG-1/2.

This coding mode also allows switching of quantizer from the one previously in use. Specification of DQUANT, a differential quantizer involves a 2-bit overhead as discussed earlier.

### 3.5.4. Bidirectional Coding

Bidirectional coding mode uses interpolated motion compensation in the same manner as in MPEG-1/2 with the difference that a VOP is used for prediction instead of a picture. Two motion vectors in half sample units are employed for a 16x16 macroblock being coded. Chrominance vectors are derived by scaling of luminance vectors as in MPEG-1/2.

This coding mode also allows switching of quantizer from the one previously in use. Specification of DQUANT, a differential quantizer involves a 2-bit overhead as discussed earlier.

### 3.5.5. Mode Decisions

Since, in B-VOPs, a macroblock can be coded in one of the four modes, we have to decide which mode is the best. At the encoder, motion compensated prediction is calculated by each of the four modes. Next, using each of the motion compensated prediction macroblocks SAD (sum of absolute differences) is computed between it and the macroblock to be coded. The MBTYPE mode is selected as follows

```
if (SADdirect - NB/2 + 1 <= min{SADinterpolate, SADbackward, SADforward})
    direct mode
else if (SADinterpolate <= min{SADbackward, SADforward, SADdirect})
    interpolate mode
else if (SADbackward <= min{SADforward, SADdirect, SADinterpolate})
    backward mode
else
    forward mode
```

### 3.5.6. Motion Vector Coding

Motion vectors are to be coded differentially. The differential motion vector coding method is same as that in MPEG-1/2. All predictions are reset at the left edge of a VOP. Depending on the macroblock type either one or both predictors may be updated, the predictors that are not updated are carried through. For macroblocks coded in direct bidirectional prediction mode, the forward and backward motion vector computed for block prediction are to be used as forward and backward motion vector predictors.

### 3.6 Rate Control

Rate control is the one used in the anchors for the November 1996 test (DOC M0322) on each VOP independently.

### 3.7 Generalized Scalable Encoding

Realizing that many applications require video to be simultaneously available for decoding at a variety of resolutions or qualities this VM supports scalability. In general, scalability of video means the ability to achieve video of more than one resolution and/or quality simultaneously. Scalable video coding involves generating a coded representation (bitstream) in a manner which facilitates the derivation of video of more than one resolution and/or quality by scalable decoding. Bitstream Scalability is the property of a bitstream that allows decoding of appropriate subsets of a bitstream to generate complete pictures of resolution and/or quality commensurate with the proportion of the bitstream decoded. If a bitstream is truly scalable, decoders of different complexities, from low performance decoders to high performance decoders can coexist, and while low performance decoders may decode only small portions of the bitstream producing basic quality, high performance decoders may decode much more and produce significantly higher quality.

Two main types of scalability are: the spatial scalability, and the temporal scalability. The spatial scalability offers scalability of the spatial resolution, and the temporal scalability offers scalability of the temporal resolution. Each type of scalability involves more than one layers. In the case of two layers consisting of a lower layer and a higher layer; the lower layer is referred to as the base-layer and the higher layer is called the enhancement-layer. Traditionally, these scalabilities are applied to frames of video such that in case of spatial scalability, the enhancement-layer frames enhances the spatial resolution of base-layer frames, while in temporal scalability, the enhancement-layer frames are temporally multiplexed with the base-layer frames to provide a higher temporal resolution video. Many MPEG-4 applications are however even more demanding and necessitate not only traditional frame based scalabilities but also scalabilities of VOPs of arbitrary shapes.

The scalability framework discussed in this VM is referred to as generalized scalability and includes the spatial and the temporal scalabilities. In the case of temporal scalability, this VM supports both frames (rectangular VOPs) as well as arbitrary shaped VOPs, however, in the case of spatial scalability, only rectangular VOPs are presently supported. Figure 3.7.1 shows a high level codec structure for generalized scalability.

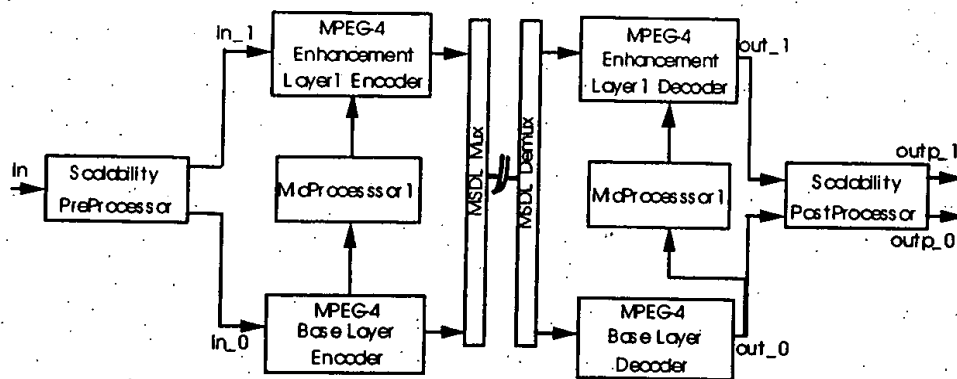


Figure 3.7.1 A High Level Codec Structure for Generalized Scalability.

Video VOPs (rectangular or otherwise) are input to Scalability PreProcessor and if spatial scalability is to be performed with base layer at lower spatial resolution and the enhancement layer at higher spatial resolution, this preprocessor performs spatial downsampling of input VOPs to generate in\_0 which forms the input to MPEG-4 Base Layer Encoder which performs non-scalable encoding. The reconstructed VOPs from base

layer are then fed to Midprocessor1 which in this case performs spatial upsampling. The other output of PreProcessor corresponds to the higher spatial layer VOPs and forms the input (in\_1) to the MPEG-4 Enhancement Layer Encoder. The base- and enhancement-layer bitstreams are multiplexed by MSDL Mux and either stored or transmitted and by employing an MSDL Demux can be retrieved for decoding by corresponding MPEG-4 Base Layer Decoder and MPEG-4 Enhancement Layer Decoder. The operation of Midprocessor1 is identical to that at the encoder. The Scalability PostProcessor performs any necessary operations such as spatial upsampling of the decoded base layer for display resulting at outp\_0 while the enhancement layer without upsampling may be output as outp\_1.

When the generalized codec is used to perform temporal scalability, the Scalability PreProcessor performs temporal demultiplexing of a VO into two substreams of VOPs, one of which (in\_0) is input to the MPEG-4 Base Layer Encoder and the other (in\_2) is input to the MPEG-4 Enhancement Layer Encoder. In this case, Midprocessor1 does not perform any spatial resolution conversion and simply allows the decoded base-layer VOPs to pass through and these VOPs are used for temporal prediction in encoding of enhancement-layer. The operation of MSDL Mux and MSDL Demux is exactly similar as in case of spatial scalability. The decoding of base and enhancement-layer bitstreams occurs in the corresponding base- and enhancement-layer decoders as shown. The PostProcessor simply outputs the base layer VOPs without any conversion, but temporally multiplexes the base and enhancement layer VOPs to produce higher temporal resolution enhancement layer.

As mentioned earlier, since VOPs can have a rectangular shape (frame) or an irregular shape, both the traditional spatial and temporal scalabilities as well as object based spatial and temporal scalabilities become possible. In this current version of the VM, spatial scalability is limited to rectangular VOPs. We now describe the encoding process for the spatial and temporal scalabilities.

### **3.7.1 Spatial Scalability Encoding**

#### **3.7.1.1 Base Layer and Enhancement Layers**

As mentioned earlier, in spatial scalability, the base layer and the enhancement layer can have different spatial resolution. In this VM, the base layer has lower resolution and the enhancement layer has higher resolution. For example, in simulations, the base-layer uses QCIF resolution and the enhancement-layer uses CIF resolution.

#### **3.7.1.2 Downsampling**

The downsampling process is performed at the scalability preprocessor. For example, the downsampling process from ITU-R 601 to CIF/QCIF is described in section 2.2.2 Filtering process. The downsampling process for the factor of 2 is only described in this document, however downsampling for arbitrary factor is allowed.

#### **3.7.1.3 Encoding of Base Layer**

The encoding process of the base layer is the same as non\_scalable encoding process.

#### **3.7.1.4 Upsampling Process**

The upsampling process is performed at the midprocessor. The VOP of the base layer is locally decoded and the decoded VOP is upsampled to the same resolution as that of the enhancement layer. In case of the example above, upsampling is performed by the filtering process described in figure 3.7.1.1 and table 3.7.1.1.

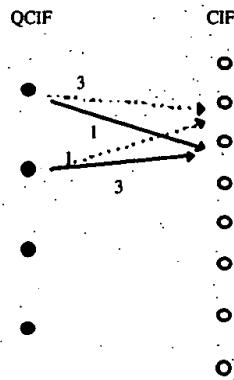


Figure 3.7.1.1

Factor	Tap no.	Filter taps	Divisor
2	1	1, 3	4
	2	3, 1	4

Table 3.7.1.1

#### 3.7.1.5. Encoding of Enhancement Layer

The VOP in the enhancement layer is encoded as either P-VOP or B-VOP. The relationship between VOP in the base layer and that of the enhancement layer is illustrated in figure 3.7.1.2. The VOP which is temporally coincident with I-VOP in the base layer is encoded as P-VOP. The VOP which is temporally coincident with P-VOP in the base layer is encoded as B-VOP. In case of the spatial scalability, a decoded VOP in the base layer is used as a reference of the prediction. The temporally coincident VOP in the reference layer (base layer) must be coded before the encoding of the VOP in the enhancement layer.

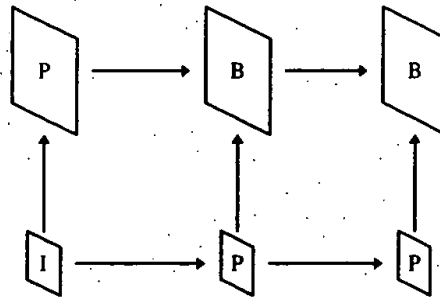


Figure 3.7.1.2

### 3.7.1.6. Encoding of P-VOPs of Enhancement Layer

In P-VOP, the `ref_select_code` is set to "11", i.e., the prediction reference is set to I-VOP which is temporally coincident VOP in the base layer.

In P-VOP, the motion vector is always set to 0, thus motion vector is not encoded to reduce the overhead.

#### Mode decisions in P-VOP

In case of P-VOP, the encoder makes a decision on whether to use INTRA or INTER prediction in the coding. The method to decide the mode is the same as INTRA/INTER mode decision in the coding of the base layer. (See sec 3.3.2.3) If INTER mode is chosen, the macroblock is coded using the prediction from the VOP in the base layer. The INTER4V mode is not used in the coding of the enhancement layer.

### 3.7.1.7. Encoding of B-VOPs of Enhancement Layer

In B\_VOP, the `ref_select_code` is set to "00", i.e., the forward prediction reference is set to P-VOP which is temporally coincident VOP in the base layer, and the backward prediction reference is set to P-VOP or B-VOP which is the most recent decoded VOP of the enhancement layer.

In B-VOP, when the forward prediction is selected, i.e., the prediction from the base layer is selected, the motion vector is always set to 0, thus motion vector is not encoded to reduce the overhead.

#### Mode decision for B-VOP

In case of the spatial scalability, the Direct (H.263 B) mode is not used. A macroblock in B-VOPs is coded in one of the other three modes. The encoder makes a decision on which mode is the best. SAD (sum of absolute differences) is calculated for each of the three modes. The MBTYPE mode is selected as follows,

```
if (SADinterpolate <= min[SADinterpolate, SADbackward, SADforward])
    interpolate mode
else if (SADbackward <= min[SADinterpolate, SADbackward, SADforward])
    backward mode
else
    forward mode
```

## 3.7.2 Temporal Scalability Encoding

In Object-based Temporal scalability (OTS), the frame rate of a selected object is enhanced such that it has a smoother motion than the remaining area. In other words, the frame rate of the selected object is higher than that of the remaining area. There are two types of enhancement structures in OTS.

Figure 3.7.2(a) shows the example of Type 1 where VOL0 (VideoObjectLayer 0) is an entire frame with both an object and a background, while VOL1 represents the particular object in VOL0. VOL0 is coded with a low frame rate and VOL1 is coded to achieve a higher frame rate than VOL0. In this example, frames 2 and 4 are formed by combining two base layer frames 0 and 6 followed by overlapping the object of the enhancement layer onto the combined frame. The combined frame is formed using a process we call "background composition", as described in Section 5.3. In this example, forward predictions forming P-VOPs are used. Figure 3.7.2(b) shows another example of Type 1 that also uses bidirectional predictions forming B-VOPs in the enhancement layer.

Figure 3.7.3 shows the example of Type 2 where VO0 (VideoObject 0) is the sequence of an entire frame which only contains a background and it has no scalability layer. VO1 is the sequence of a particular object and it has two scalability layers, VOL0 and VOL1. VOL1 represents the same object as VOL0 and it is

coded to achieve a higher frame rate than VOL0. In this example, VOL0 is regarded as a base layer and VOL1 is regarded as an enhancement layer of the OTS. Note that the VOL0 may not have the same frame rate as other VOs.

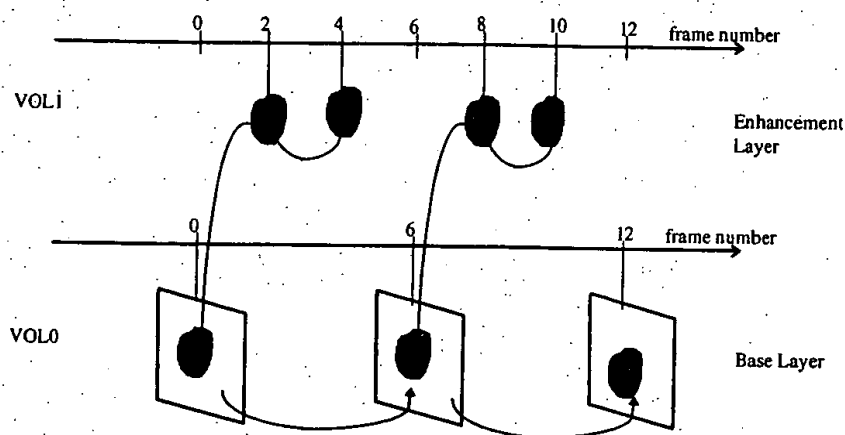


Figure 3.7.2(a) : Enhancement structure of Type 1 with P-VOPs.

Comment: PAGE 49

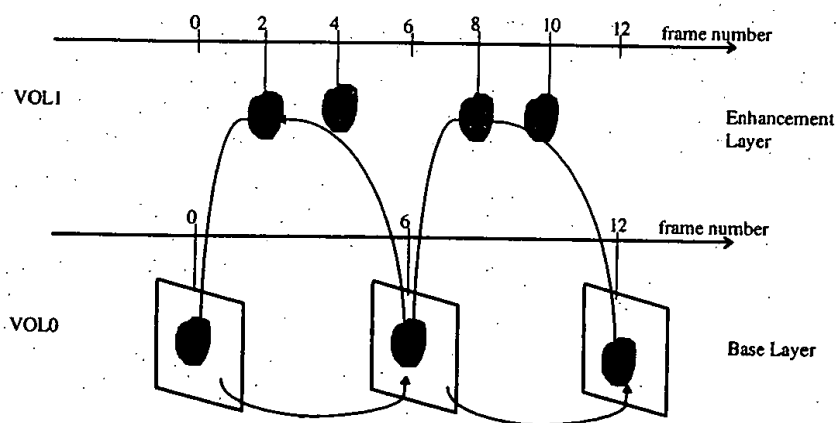


Figure 3.7.2(b) : Enhancement structure of Type 1 with B-VOPs.

Comment: PAGE 49



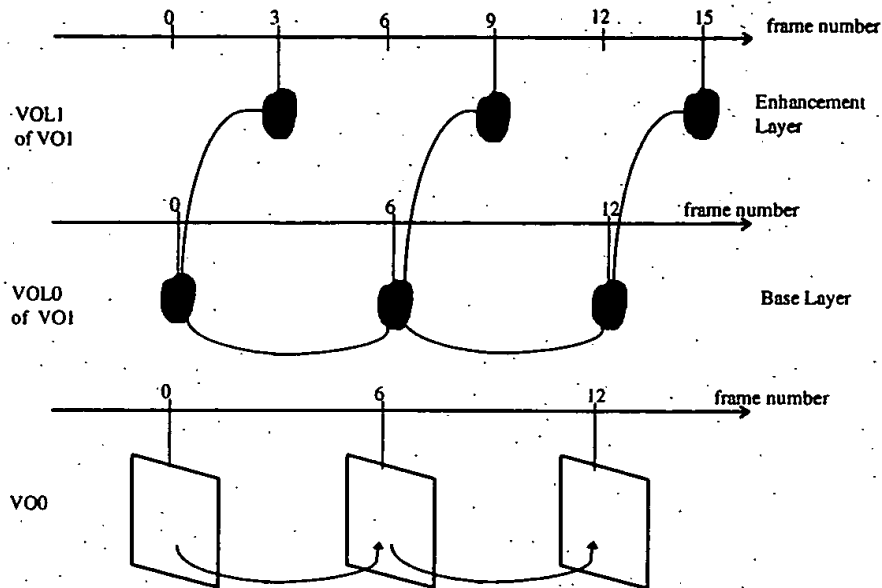
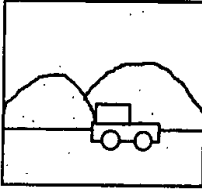
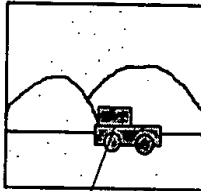
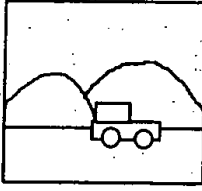
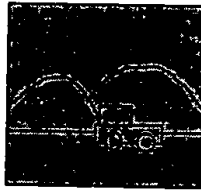




Figure 3.7.3 : Enhancement structure of Type 2.

There are 2 types of enhancements for scalability, described by the `enhancement_type` flag. We explain below the meaning of `enhancement_type` flag in more detail. As an example, Figure 3.7.4 shows an entire image containing several types of regions; for example a road, a car, and mountains. Both the base layer with `enhancement_type` being "0" and the base layer with `enhancement_type` being "1" are coded with lower picture quality which means that either the frame rate is lower or the spatial resolution is lower. At the enhancement layer of the scalability, `enhancement_type` flag distinguishes the following two cases.

- When this flag is "1", the enhancement layer increases the picture quality of a partial region of the base layer. For example, in Figure 3.7.4, VOL0 is an entire frame and VOL1 is the car in the frame. The temporal resolution or the spatial resolution of the car is enhanced.
- When this flag is "0", the enhancement layer increases the picture quality of the entire region of the base layer. For example, in Figure 3.7.4, if VOL0 represents an entire frame, VOL1 is also the entire frame. Then the temporal or spatial resolution of entire frame is enhanced. If VOL0 represents the car, VOL1 is also the car which is enhanced in terms of temporal or spatial resolution.

Note that since only rectangular VOP-based spatial scalability is included in this current version of the VM, `enhancement_type` flag is always set to "0" for spatial scalability.

	Base layer	Enhancement layer
enhancement_type = 1	 VOL0 : entire frame	 VOL1 : car
enhancement_type = 0	 VOL0 : entire frame	 VOL1 : entire frame
	 VOL0 : car	 VOL1 : car

 : region to be enhanced by an enhancement layer

Figure 3.7.4 : Example of a region to be enhanced.

## 4. Bitstream Syntax

### 4.1 General Structure

The syntax consists of the following class hierarchy:

- VideoSession (VS)

- VideoObject (VO)
- VideoObjectLayer (VOL)
- VideoObjectPlane (VOP)

Within the context of video experiments, a VS is a collection of one or more VO's, a VO can consist of one or more layers and that each layer consists of an ordered sequence of snapshots in time called VOPs. Thus there can be several VO's (VO0, VO1,...) in a VS and for each VO, there can be several layers (VOL0, VOL1,...) and each layer consists of time sequence of VOPs (VOP0, VOP1,...); which are basically snapshots in time. A VO can be of arbitrary shape (rectangular is a special case). For single layered coding only one VOL (VOL0) exists per VO. Figure xxx shows the hierarchical structure of the syntax.

For the purpose of conducting core experiments the bitstreams for the VideoSession and each of the VideoObject's are stored in separate files. The multiplexing of these bitstreams will be provided by the MSDL.

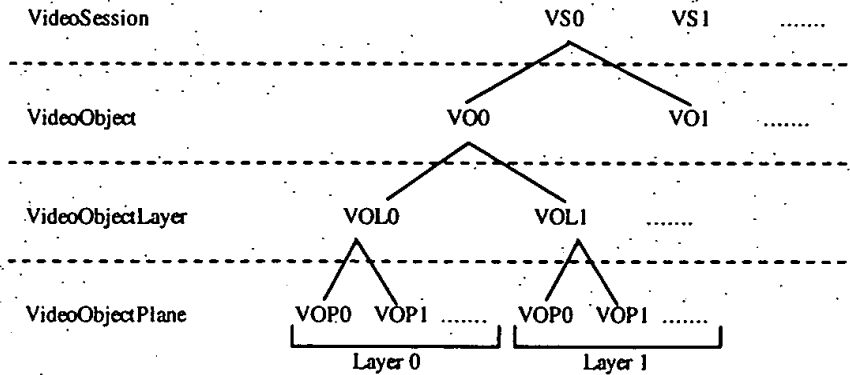


Figure xxx : Hierarchy in the proposed video syntax

## 4.2 Video Session Class

### Video Session Class

Syntax	No. of bits	Mnemonic
VideoSession() {		
video_session_start_code	sc+8 =32	
do* {		
VideoObject()		
} while (nextbits() == video_object_start_code)		
video_session_end_code	sc+8 = 32	
}		

\* concurrent loop solution to be provided by MSDL.

sc

This code is intended to be used in combination with additional bits, for the purpose of the synchronization. Its binary representation is 23 zeros followed by a 1 (000000000000000000000001), or its hexadecimal representation is '000001'.

video\_session\_start\_code

**video\_session\_end\_code**

### 4.3. Video Object Class

Syntax	No. of bits	Mnemonic
VideoObject() { video_object_start_code video_object_id do { VideoObjectLayer() } while (nextbits() == video_object_layer_start_code) }	sc+3=27 5	

**video\_object\_id**

52

## 4.4 Video Object Layer Class

### Video Object Layer

Syntax	No. of bits	Mnemonic
VideoObjectLayer() {		
video_object_layer_start_code	sc+4=28	
video_object_layer_id	4	
video_object_layer_shape	2	
if ( video_object_layer_shape == '00' ) {		
video_object_layer_width	10	
video_object_layer_height	10	
}		
video_object_layer_quant_type	1	
if (video_object_layer_quant_type) {		
load_intra_quant_mat	1	
if (load_intra_quant_mat)		
intra_quant_mat[64]	8*64	
load_nonintra_quant_mat	1	
if (load_nonintra_quant_mat)		
nonintra_quant_mat[64]	8*64	
}		
intra_dcprd_disable	1	
video_object_layer_fcode_forward	2	
video_object_layer_fcode_backward	2	
separate_motion_shape_texture	1	
scalability	1	
if (scalability) {		
ref_layer_id	4	
ref_layer_sampling_direct	1	
hor_sampling_factor_n	5	
hor_sampling_factor_m	5	
vert_sampling_factor_n	5	
vert_sampling_factor_m	5	
enhancement_type	1	
}		
do {		
VideoObjectPlane()		
} while (nextbits() == video_object_plane_start_code)		
}		

**video\_object\_layer\_start\_code**

This is a unique code of length 28 bits (sc+4) that precedes the video\_object\_layer\_id.

**video\_object\_layer\_id**

This is a 4-bit code which identifies a video object layer for a video object being processed.

**video\_object\_layer\_shape**

This is a 2-bit code which identifies the shape type of video object layer as shown in Table 0x.

**Table 0x: Video Object Layer shape types**

video_object_layer_shape	Code
rectangular	00

binary	01
gray-scale	10

This flag is "00" if a VOL shape is rectangular, "01" if a VOL has a binary shape (i.e. if each pixel of the rectangle defined by *video\_object\_layer\_width* and *video\_object\_layer\_height* is either part of a VOL or not), and "10" if a VOL shape is defined by grey scale data (i.e. if each pixel of the rectangle defined by *video\_object\_layer\_width* and *video\_object\_layer\_height* is to be linearly combined with the pixels of other VOLs at the same spatial location).

#### **video\_object\_layer\_width, video\_object\_layer\_height**

These two codes define the picture size for the session, in pixels unit (zero values are forbidden). This is also the size of the unique VOL of the session.

#### **video\_object\_layer\_quant\_type**

A 1-bit code which indicates the type of quantization method selected. When it has a value of 0, H.263 quantization method is selected, otherwise, MPEG-1/2 quantization method is selected.

#### **load\_intra\_quant\_mat**

A 1-bit code which indicates whether the default matrix for visually weighting DCT coefficients of intra macroblocks is selected or if a new matrix for visually weighting of DCT coefficients of intra macroblocks is to be loaded.

#### **intra\_quant\_mat[64]**

This is an one dimensional (1d) array of 64 values (8-bits per value expressed in range 1 to 255) for visual weighting of DCT coefficients of intra macroblocks. The values in the 1d array are in the same order as that obtained by zig-zag scanning of a 8x8 two-dimensional array. In addition, the first value for *intra\_quant\_mat[64]* should always be 8.

#### **load\_nonintra\_quant\_mat**

A 1 bit code which indicates whether the default matrix for visually weighting DCT coefficients of nonintra macroblocks is selected or if a new matrix for visually weighting of DCT coefficients of nonintra macroblocks is to be loaded.

#### **nonintra\_quant\_mat[64]**

This is an one dimensional (1d) array of 64 values (8-bits per value expressed in range 0 to 255) for visual weighting of DCT coefficients of nonintra macroblocks. The values in the 1d array are in the same order as that obtained by zig-zag scanning of a 8x8 two-dimensional array.

#### **intra\_dc\_pred\_disable**

A 1 bit code whose value is 1 when DC prediction of intra coded blocks is to be disabled. In default mode, dc prediction of intra macroblocks is enabled.

#### **video\_object\_layer\_fcode\_forward, video\_object\_layer\_fcode\_backward**

These are 2-bit codes that specify the dynamic range of motion vectors.

#### **separate\_motion\_shape\_texture**

This flag is "1" if all the coding data (e.g. motion, shape, texture, etc) for the VOP are grouped together. It is "0" if the coding data are grouped macroblock per macroblock.

#### **scalability**

This is a 1-bit flag which indicates if the current layer uses scalable coding. If the current layer is used as the base-layer, this flag is '0'.

#### **ref\_layer\_id**

This is a 4-bit code which indicates the layer to be used as reference for the prediction(s) in the case of scalability. It can have a value between 0 and 15.

**ref\_layer\_sampling\_dirac**

This is a 1-bit flag whose value when "0" indicates that the reference layer specified by ref\_layer\_id has the same or lower resolution as the layer being coded. Alternatively, a value of "1" indicates that the resolution of reference layer is higher than the resolution of layer being coded resolution.

**hor\_sampling\_factor\_n, hor\_sampling\_factor\_m**

These are 5-bit quantities in range 1 to 31 whose ratio hor\_sampling\_factor\_n/hor\_sampling\_factor\_m indicates the resampling needed in horizontal direction; the direction of sampling is indicated by ref\_layer\_sampling\_dirac.

**vert\_sampling\_factor\_n, vert\_sampling\_factor\_m**

These are 5-bit quantities in range of 1 to 31 whose ratio vert\_sampling\_factor\_n/vert\_sampling\_factor\_m indicates the resampling needed in vertical direction; the direction of sampling is indicated by ref\_layer\_sampling\_dirac.

**enhancement\_type**

This is a 1-bit flag which indicates the type of an enhancement structure in a scalability. It has a value of "1" when an enhancement layer enhances a partial region of the base layer. It has a value of "0" when an enhancement layer enhances entire region of the base layer. The default value of this flag is "0".

## 4.5 VideoObjectPlane Class

### VideoObjectPlane

Syntax	No. of bits	Mnemonic
VideoObjectPlane() {		
VOP_start_code	sc+8=32	
do {		
modulo_time_base	1	
} while ( modulo_time_base != "0")		
VOP_time_increment	10	
VOP_prediction_type	2	
if (video_object_layer_shape != "00") {		
VOP_width	10	
VOP_height	10	
VOP_horizontal_mc_spatial_ref	10	
marker_bit	1	
VOP_vertical_mc_spatial_ref	10	
if (scalability && enhancement_type)		
background_composition	1	
}		
if (VOP_prediction_type=="10")		
VOP_dbquant	2	
else		
VOP_quant	5	
if (!scalability) {		
if (!separate_motion_shape_texture)		
combined_motion_shape_texture_coding()		
else {		
do {		
first_MMR_code	1-2	
} while (count of macroblocks != total number of macroblocks)		
motion_coding()		
shape_coding()		
texture_coding()		
}		
}		
else {		
if (background_composition) {		
load_backward_shape	1	
if (load_backward_shape) {		
backward_shape_coding()		
load_forward_shape	1	
if (load_forward_shape)		
forward_shape_coding()		
}		
}		
ref_select_code	2	
if (VOP_prediction_type=="01"    VOP_prediction_type=="10") {		
forward_temporal_ref	10	
if (VOP_prediction_type == "10") {		
marker_bit	1	
backward_temporal_ref	10	
}		
}		



combined\_motion\_shape\_texture\_coding()

#### VOP\_start\_code

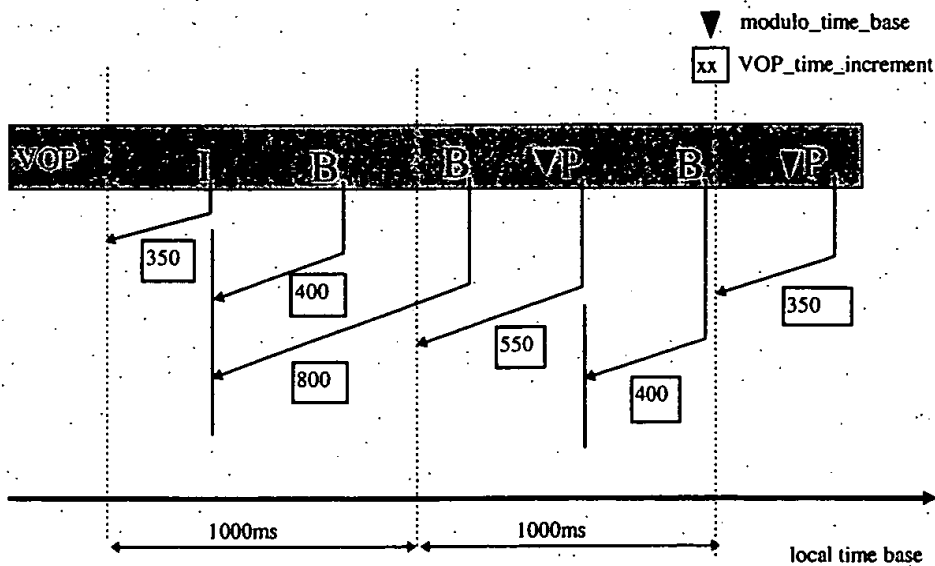
This code cannot be emulated by any combination of other valid bits in the bitstream, and is used for synchronization purpose. It is se followed by a unique 8-bit code.

#### modulo\_time\_base

This value represents the local time base at the one second resolution unit (1000 milliseconds). It is represented as a marker transmitted in the VOP header. The number of consecutive "1" followed by a "0" indicates the number of seconds has elapsed since the synchronisation point marked by the last encoded/decoded modulo\_time\_base.

#### VOP\_time\_increment

This value represents the local time base in the units of milliseconds. For I and P-VOP's this value is the absolute VOP\_time\_increment from the synchronisation point marked by the last modulo\_time\_base. For the B-VOP's this value is the relative VOP\_time\_increment from the last encoded/decoded I- or P-VOP.



To produce a picture at a given time (according to the display frame rate), the simplest solution is to use the most recently decoded data of each VOP to be displayed. Another possibility, more complex and for non real time applications, could be to interpolate each VOP from its two occurrences temporally surrounding the needed instant, based on their temporal references.

#### VOP\_prediction\_type

This code indicates the prediction mode to be used for decoding the VOP as shown in Table 1x.

Table 1x: VOP prediction types

VOP_prediction_type	Code
---------------------	------

I	00
P	01
B	10

#### **VOP\_width, VOP\_height**

These two numerical values define the size of the smallest rectangle that includes the VOP, in pixels unit (zero values are forbidden).

#### **VOP\_horizontal\_mc\_spatial\_ref, VOP\_vertical\_mc\_spatial\_ref**

These values are used for decoding and for picture composition. They indicate the spatial position of the top left of the rectangle defined by *VOP\_width* and *VOP\_height*, in pixels unit, the origin of coordinates being the top-left corner of the picture. The corresponding values are binary represented as positive integers. This is a script information, that may be changed under user request.

#### **marker\_bit**

This is a single bit always set to '1' in order to avoid start code emulation.

#### **background\_composition**

This flag only occurs when scalability flag has a value of "1". The default value of this flag is "0". This flag is used in conjunction with *enhancement\_type* flag. If *enhancement\_type* is "1" and this flag is "1", background composition is performed. If *enhancement\_type* is "1" and this flag is "0", background is repeated from the nearest frame in base layer. Further, if *enhancement\_type* is "0" no action needs to be taken as a consequence of any value of this flag.

#### **shape\_coding()**

The *shape\_coding()* function generates the format of the coded data of a current shape (alpha plane).

#### **load\_backward\_shape**

If this flag is "1", *backward\_shape* of the previous VOP is copied to *forward\_shape* for the current VOP and *backward\_shape* for the current VOP is decoded from the bitstream. If not, *forward\_shape* for the previous VOP is copied to *forward\_shape* for the current VOP and *backward\_shape* for the previous VOP is copied to *backward\_shape* for the current VOP.

#### **backward\_shape\_coding()**

It specifies the format of coded data for *backward\_shape* and is identical to that of *shape\_coding()*.

#### **load\_forward\_shape**

This flag is "1" if *forward\_shape* will be decoded from a bitstream.

#### **forward\_shape\_coding()**

It specifies the format of coded data for *forward\_shape* and is identical to that of *shape\_coding()*.

#### **ref\_select\_code**

This is a 2-bit code which indicates prediction reference choices for P- and B-VOPs in the enhancement layer with respect to decoded reference layer identified by *ref\_layer\_id*.

#### **forward\_temporal\_ref**

An unsigned integer value which indicates temporal reference of the decoded reference layer VOP to be used for forward prediction (Table 5.3.1 and 5.3.2)

#### **backward\_temporal\_ref**

An unsigned integer value which indicates temporal reference of the decoded reference layer VOP to be used for backward prediction (Table 5.3.2).

#### **VOP\_dbquant**

VOP\_dbquant is present if VOP\_prediction\_type indicates VOP\_prediction\_type='10'. dquant ranges from 1 to 31. VOP\_dbquant is a 2-bit fixed length code that indicates the relationship between quant and bquant. Depending on the value of VOP\_dbquant, bquant is calculated according to the relationship shown in Table 2x and is clipped to lie in the range 1 to 31. In this table "/" means truncation.

**Table 2x: VOP\_dbquant codes and relation between quant and bquant**

dbquant	bquant
00	$(5 \times \text{quant})/4$
01	$(6 \times \text{quant})/4$
10	$(7 \times \text{quant})/4$
11	$(8 \times \text{quant})/4$

#### **VOP\_quant**

A fixed length codeword of 5 bits which indicates the quantizer to be used for VOP until updated by any subsequent value DQUANT. The codewords are natural binary representations of the value of quantization which being half the step sizes range from 1 to 31.

**first\_MMR\_code** For I-VOP: '00' indicates that the subsequent alpha data exist (multilevel). '01' indicates that all samples in the alpha block are "0" and '1' indicates that all samples are "255". For P-BOP and B-VOP: '00' indicates that the subsequent alpha data exist (multilevel). '01' indicates that all samples in the alpha block are "1" and '10' indicates that all samples are "255". '11' indicates that alpha data is inter-coded..

## **4.6 Shape coding**

### **shape coding**

Syntax	No. of bits	Mnemonic
<pre> shape_coding() {   if (video_object_shape != '00') {     do {       if (first_MMR_code=="00") {         CR         a0_color         do {           VLC_binary           if( Mode==H ) {             if( Vertical_pass_mode == TRUE ) {               RLB             } else {               ULB             }           } while( Mode != EOMB )         }       } while( count of macroblock != total number of macroblocks)     } if (video_object_shape == '10')     do {       Gray_shape_coding()     } while( count of macroblock != total number of macroblocks)   } } </pre>	<p>1-2</p> <p>1</p> <p>1-9</p> <p>2-4</p> <p>2-4</p>	

CR -- Conversion ratio described in section 3.2.1. The codeword table is shown in table M1.

Table M1 VLC for CR

CR	Code
1	0
1/2	10
1/4	11

a0\_color -- A 1 bit code indicating the color of the first pixel in a MB (0:white, 1:black).

VLC\_binary -- Variable length code for binary shape information shown in Table M2.

Table M2 VLC table for Modified MMR

Mode	Code
V0(DIST=0)	1
V1(DIST=1)	01s
V2(DIST=2)	00001s
V3(DIST=3)	000001s
V4(DIST=4)	0000001s
V5(DIST=5)	00000001s
H	001
EOMB	0001

V0 - V5 : Vertical mode

H : Horizontal mode

EOMB : End of MB

DIST : absolute value of ( $r_{a1} - r_{b1}$ )

s : sign bit (s=1 if  $r_{a1} - r_{b1} > 0$ , and s=0 if  $r_{a1} - r_{b1} < 0$ )

**Vertical\_pass\_mode** – TRUE=The mode is Vertical Pass mode.

FALSE=The mode is not Vertical Pass mode (it means  
Horizontal mode or Vertical mode).

**RLB** –Residual-length of binary shape information.

**ULB** – Unchanged-length of binary shape information.

## 4.7 Motion Shape Texture

### 4.7.1 Combined Motion Shape Texture

The motion shape texture coding method used for I-, P- and B-VOPs is described in Appendix B. The advanced prediction mode and overlapping motion estimation are also described in that Appendix. The macroblock layer syntax for each coded macroblock consists of macroblock header information which also includes motion vectors, and, block data which consists of DCT data (coded texture information).

### 4.7.2 Separate Motion Shape Texture for I- and P-VOPs

#### motion\_coding

The syntax for the encoded motion vectors of macroblocks that belong to the VOP would be:

1 - 2	N bits	1 - 2	N bits
No. of Vectors	Encoded vector ...	No. of Vectors	Encoded vector .... ...

#### No. of Vectors:

Huffman coded number of motion vectors for each macroblock (0, 1, or 4).

A '0' indicates that there is no motion compensation for macroblock. Hence the data coded by the texture coding part will be the actual pixel values in the current image at this macroblock location (INTRA coded) or skipped by the texture coding syntax. A '1' indicates that this macroblock is motion compensated by a 16 x 16 motion vector. Similarly a '4' indicates that this macroblock is motion compensated by 4, 8 x 8 motion vectors.

The Huffman codes used to code this No. of Vectors field are

Value	Length	Code
0	2	11
1	1	0
4	2	10

#### Encoded vector:

These are coded differentially coded using the same prediction scheme and Huffman tables, as described in in sections 3.3.2.6 to 3.3.2.8.

#### shape\_coding

As in section 4.6.

#### texture\_coding

The texture data for the macroblocks belonging to the VOP is coded using a DCT coding as in section 3.4. The syntax of the texture coding for each of the macroblocks in the VOP is as follows:

1            2-6    2-3        N    bits

DCT/NO_DCT flag	CBPY	CBPC	DCT data
-----------------	------	------	----------

#### DCT/NO\_DCT flag:

This flag indicates whether a macroblock had DCT data or not. If it has DCT data. The Huffman Codes used to code the DCT/NO\_DCT flag are:

Value	Length	Code
DCT	1	0
NO_DCT	1	1

To skip a macroblock, No. Of Motion Vectors is set to '00' and the DCT/NO\_DCT flag to '1'.

#### CBPY:

Coded Block Pattern Luminance, uses the same Huffman tables as in appendix B.

#### CBPC:

Coded Block Pattern Chrominance, uses the following Huffman table

Value	Length	Code
00	1	1
01	3	001
10	3	010
11	3	011

#### DCT data:

DCT encoded macroblock data using the same 3D VLC as in appendix B.

### **4.7.3 Separate Motion Shape Texture for B-VOPs**

#### **motion\_coding**

The syntax for the encoded macroblock overhead information and motion vectors of macroblocks that belong to B-VOP is:

MB header MB header ... MB header MB header ....

As defined earlier in section 4.7.1, the MB header consists of MODB, MBTYPE, CBPB, DQUANT and any associated motion vector/s (as indicated by MBTYPE).

#### **shape\_coding**

As in section 4.6

#### **texture\_coding**

The syntax for the encoded DCT coefficients representing texture data is as follows:

```
Block Data | .. Block Data | Block Data .... .. |
```

Again as defined in section 4.7.1, the Block data simply consists of DCT coefficients.

## 5. Decoder Definition

### 5.1 Overview

The Figure 5.1.1 presents a general overview of the VOP decoder structure. The same decoding scheme is applied when decoding all the VOPs of a given session.

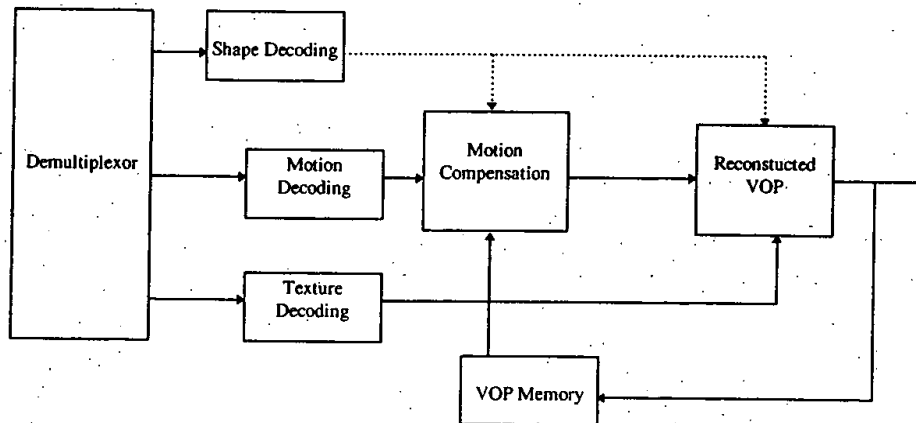


Figure 5.1.1 : VOP decoder structure.

The decoder is mainly composed of two parts : the shape decoder and the traditional motion & texture decoder. The reconstructed VOP is obtained by the right combination of the shape, texture and motion information.

### 5.2 Shape decoding

Decoding process is described below.

```

abs_a0 = 0
DECODE a0_color
DETECT b1
if( b1 is detected.)
    Vertical_pass_mode = FALSE
else

```



```

Vertical_pass_mode = TRUE
do {
  if( Mode == H ) {
    if( Vertical_pass_mode == FALSE ) {
      ( see 3.2.1.3.1 )
      DECODE ULB
      if( ULB == WIDTH ) Vertical_pass_mode = TRUE
      else      abs_a0 = abs_a1
    } else { /* Vertical_pass_mode == TRUE */
      ( see 3.2.1.3.2 )
      DECODE RLB
      Vertical_pass_mode = FALSE
      abs_a0 = abs_a1
    }
  }
  else if ( Mode == V0 ) {
    if( Vertical_pass_mode == TRUE ) { ( see 3.2.13.2 ) }
    else { DETECT b1
           r_a1 = r_b1
           abs_a0 = abs_a1 }
  }
  else if ( Mode == V1 ) {
    DETECT b1
    DIST = 1
    if ( s == 0 ) r_a1 = r_b1 - DIST
    else      r_a1 = r_b1 + DIST
    abs_a0 = abs_a1
  }
  else if ( Mode == V5 ) {
    DETECT b1
    DIST = 5
    if ( s == 0 ) r_a1 = r_b1 - DIST
    else      r_a1 = r_b1 + DIST
    abs_a0 = abs_a1
  }
} while( Mode != EOMB )

```

### 5.3 Generalized Scalable Decoding

We now discuss the decoding issues in generalized scalable decoding. Considering the case of two layers, a base-layer and an enhancement-layer, the spatial resolution of each layer may be either the same or different; when the layers have different spatial resolution, (up or down) sampling of base-layer with respect to the enhancement-layer becomes necessary for generating predictions. If the lower layer and the enhancement-layer are temporally offset, irrespective of the spatial resolutions, motion compensated

prediction may be used between layers. When the layers are coincident in time but at different resolution, motion compensation may be switched off to reduce overhead.

The reference VOPs for prediction are selected by `reference_select_code` as specified in Tables 5.3.1 and 5.3.2. In coding of P-VOPs belonging to an enhancement layer, the forward reference is one of the following three: the most recent decoded VOP of enhancement layer, the most recent VOP of the lower layer in display order, or the next VOP of the lower layer in display order.

In B-VOPs, the forward reference is one of the two: the most recent decoded enhancement VOP or the most recent lower layer VOP in display order. The backward reference is one of the three: the temporally coincident VOP in the lower layer, the most recent lower layer VOP in display order, or the next lower layer VOP in display order.

*Table 5.3.1 : Prediction reference choices for P-VOPs in the object-based temporal scalability.*

ref_select_code	forward prediction reference
00	Most recent decoded enhancement VOP belonging to the same layer.
01	Most recent VOP in display order belonging to the reference layer.
10	Next VOP in display order belonging to the reference layer.
11	Temporally coincident VOP in the reference layer (no motion vectors)

*Table 5.3.2 : Prediction reference choices for B-VOPs in the case of scalability.*

ref_select_code	forward temporal reference	backward temporal reference
00	Temporally coincident VOP in the reference layer (no motion vectors)	Most recent decoded enhancement VOP of the same layer
01	Most recent decoded enhancement VOP of the same layer.	Most recent VOP in display order belonging to the reference layer.
10	Most recent decoded enhancement VOP of the same layer.	Next VOP in display order belonging to the reference layer.
11	Most recent VOP in display order belonging to the reference layer.	Next VOP in display order belonging to the reference layer.

The enhancement-layer can contain I-, P- or B-VOPs, but the B-VOPs in the enhancement layer behave more like P-VOPs at least in the sense that a decoded B-VOP can be used to predict the following P- or B-VOPs.

When the most recent VOP in the base layer is used as reference, this includes the VOP that is temporally coincident with the VOP in the enhancement layer. However, this necessitates use of the base layer for motion compensation which requires motion vectors.

If the coincident VOP in the lower layer is used explicitly as reference, no motion vectors are sent and this mode can be used to provide spatial scalability. Spatial scalability in MPEG-2 uses spatio-temporal prediction, which is accomplished here by using the prediction modes available for B-VOPs.

### **5.3.1 Spatial Scalability Decoding**

#### **5.3.1.1. Base Layer and Enhancement Layer**

For spatial scalability, the output from decoding the base layer only have different spatial resolution from output of decoding both the base layer and the enhancement layer. For example, the resolution of the base layer is QCIF resolution and that of the enhancement layer is CIF resolution. In this case, when the output with QCIF resolution is required, only the base layer is decoded. And when the output with CIF resolution is required, both the base layer and the enhancement layer are decoded.

#### 5.3.1.2. Decoding of Base Layer

The decoding process of the base layer is the same as non\_scalable decoding process.

#### 5.3.1.3. Upsampling Process

The upsampling process is performed at the midprocessor. The VOP of the base layer is locally decoded and the decoded VOP is upsampled to the same resolution as that of the enhancement layer. In case of the example above, upsampling is performed by the filtering process described in Figure 3.7.1.1 and Table 3.7.1.1.

#### 5.3.1.4. Decoding Process of Enhancement Layer

The VOP in the enhancement layer is decoded as either P-VOP or B-VOP.

#### 5.3.1.5. Decoding of P-VOPs in Enhancement Layer

In P-VOP, the ref\_select\_code is always "11", i.e., the prediction reference is set to I-VOP which is temporally coincident VOP in the base layer. In P-VOP, the motion vector is always set to 0 at the decoding process.

#### 5.3.1.6. Decoding of B-VOPs in Enhancement Layer

In B-VOP, the ref\_select\_code is always "00", i.e., the forward prediction reference is set to P-VOP which is temporally coincident VOP in the base layer, and the backward prediction reference is set to P-VOP or B-VOP which is the most recent decoded VOP of the enhancement layer. In B-VOP, when the forward prediction, i.e., the prediction from the base layer is selected, the motion vector is always set to 0 at the decoding process.

### 5.3.2 Temporal Scalability Decoding

In object based temporal scalability, a background composition technique is used in the case of Type 1 scalability as discussed in Section 3.7.2. Background composition is used in forming the background region for objects at the enhancement layer. We now describe the background composition technique referring to Figure 5.3.1, where background composition for a current VOP is depicted, where composition is performed using the previous and the next pictures in the base layer (e.g., the background region for the VOP at frame 2 in Figure 3.7.2 is composed using frames 0 and 6 in the base layer).

In Figure 5.3.1, we show the background composition for the current frame at the enhancement layer. The dotted line represents the shape of the selected object at the previous frame in the base layer (called "forward shape"). As the object moves, its shape at the next frame in the base layer is represented by a broken line (called "backward shape"). For the region where the areas enclosed by these shapes overlap, the pixel value from the nearest frame at the base layer is used for the composed frame. Similarly, outside these objects, the pixel value from the nearest frame at the base layer is used. These areas are shown as white in Figure 5.3.1. For the region occupied by only the selected object of the previous frame at the base layer, the pixel value from the next frame at the base layer is used for the composed frame. This area is shown as lightly shaded in Figure 5.3.1. On the other hand, for the region occupied by only the selected object of the next frame at the base layer, pixel values from the previous frame are used. This area is darkly shaded in Figure 5.3.1.

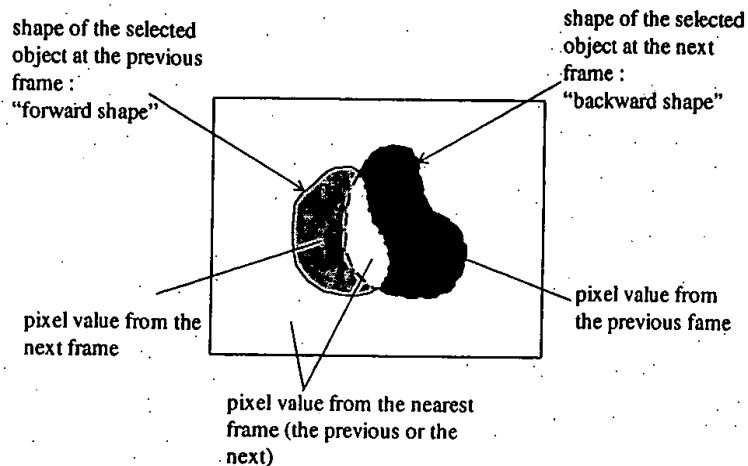


Figure 5.3.1: Background composition.

The following process is a mathematical description of the background composition method.

If (  $s(x, y, ta)=1$  and  $s(x, y, td)=1$  ) or (  $s(x, y, ta)=0$  and  $s(x, y, td)=0$  ),

$$fc(x, y, t) = f(x, y, td) \quad (|t - ta| > |t - td|)$$

$$fc(x, y, t) = f(x, y, ta) \quad (\text{otherwise}),$$

if  $s(x, y, ta)=1$  and  $s(x, y, td)=0$ ,

$$fc(x, y, t) = f(x, y, td),$$

if  $s(x, y, ta)=0$  and  $s(x, y, td)=1$ ,

$$fc(x, y, t) = f(x, y, ta),$$

where

$fc$  : composed image  
 $f$  : decoded image of the base layer  
 $s$  : shape information(alpha plane)  
 $(x, y)$  : the spatial coordinate  
 $t$  : time of the current frame  
 $ta$  : time of the previous frame  
 $td$  : time of the next frame

Two types of shape information,  $s(x, y, ta)$  and  $s(x, y, td)$ , are necessary for the background composition.  $s(x, y, ta)$  is called a "forward shape" and  $s(x, y, td)$  is called a "backward shape" in Section 4.4. When a gray scale alpha plane is used, positive value is regarded as the value "1" of the binary alpha plane. Note that the above technique is based on the assumption that the background is not moving.

## 5.4 Compositor Definition

The output of the decoders are the reconstructed VOP's that are passed to the compositor. In the compositor the VOP's are recursively blended in the order specified by the VOP\_composition\_order.

Each VOP has its own YUV and alpha values. Blending is done sequentially ( two layers at a time). For example, if VOP N is overlayed over VOP M to generate a new VOP P, the composited Y, U, V and alpha values are:

$$Pyuv = ((255 - Nalpha) * Myuv + (Nalpha * Nyuv ))/255;$$

$$Palpha = 255.$$

In the case that there exists more than two VOPs for a particular sequence, this blending procedure recursively applies to YUV components by taking the output picture as background.

## 5.5 Flex\_0 Composition Layer Syntax

A composition script describes the arrangement of AV objects in a scene. In Flex\_1, this composition script is expressed in a procedural language (such as Java). In Flex\_0, this composition script is expressed instead by a fixed set of parameters. That is, the composition script is parametrized. These parameters are encoded and transmitted in a composition layer. This section briefly describes the bitstream syntax of the composition layer for Flex\_0.

### 5.5.1 Bitstream Syntax

At any given time, a scene is composed of a collection of objects, according to composition parameters. The composition parameters for an object may be changed at any time by transmitting new parameters for the object. These parameters are timestamped, in order to be transmitted only occasionally if desired.

```
class SessionParameters {
    while (! [end_of_session])
        uint(30) timestamp; // millisec since last update
        CompositionInformation composition_information;
};

map motion_sets_table( int ) {
    0b0,      0,
    0b10,     1,
    0b110,    2,
    0b1110,   3,
    0b1111,   4
};

class CompositionInformation {
    uint(5)   video_object_id;
    bit(1)    visibility;
    if (visibility) {
```

```

    bit(1)    3_dimensional;
    if (!3_dimensional)
        uint(5)    composition_order;
    vlc(motion_sets_table) number_of_motion_sets;
    if (number_of_motion_sets > 0) {
        int(10)    x_translation;
        int(10)    y_translation;
        if (3_dimensional)
            int(10)    z_translation;
    }
    if (number_of_motion_sets > 1) {
        int(10)    x_delta_1;
        int(10)    y_delta_1;
        if (3_dimensional)
            int(10)    z_delta_1;
    }
    if (number_of_motion_sets > 2) {
        int(10)    x_delta_2;
        int(10)    y_delta_2;
        if (3_dimensional)
            int(10)    z_delta_2;
    }
    if (number_of_motion_sets > 3) {
        int(10)    x_delta_3;
        int(10)    y_delta_3;
        if (3_dimensional)
            int(10)    z_delta_3;
    }
}
};

```

### 5.5.2 Parameter Semantics

The meaning of the above parameters are:

**uint(5) video\_object\_id**

The ID of the VO whose composition information this data represents.

**boolean visibility**

Set if the object is visible.

**boolean 3\_dimensional**

If set the object has 3D extent else it purely is 2D.

**uint(3) number\_of\_motion\_sets**

Number of X,Y (and Z) data sets provided for translation and rotation.

**uint(5) composition\_order**

This field is used to indicate the place in the object stack this object should be visualised. It is used to determine which objects occlude which other objects for 2D composition.

**int(10) x\_translation, y\_translation, z\_translation**

Translation of the object relative to the origin of the scene coordinate system.

**int(10) x\_delta\_n, y\_delta\_n, z\_delta\_n**

Coordinate transformation information as per input contribution ml119.doc

## Appendix A

### Combined Motion Shape Texture Coding

This appendix describes the combined motion shape texture coding. The text is based on Draft ITU-T Recommendation H.263 Video Coding for Low Bitrate Communication sections 5.3 and 5.4. Annexes describing optional modes can be found in that document.

#### B.1. Macroblock Layer

Data for each macroblock consists of a macroblock header followed by data for blocks. The macroblock layer structure in I or P VOPs is shown in Figure B.2(a). First\_MMR\_code is only present for which VOP of arbitrary shape is '1'. COD is only present in VOPs for which VOP\_prediction\_type indicates P-VOPs (VOP\_prediction\_type == '01'). MCBPC is present when indicated by COD or when VOP\_prediction\_type indicates I-VOP (VOP\_prediction\_type == '00'). CBPY, DQUANT, and MVD<sub>2,4</sub> are present when indicated by MCBPC. Block Data is present when indicated by MCBPC and CBPY. MVD<sub>2,4</sub> are only present in Advanced Prediction mode. CR, a0\_color, VLC\_binary, RLB, ULB are only present when first\_MMR\_code indicates multilevel (see sec. 4.6).

first_MMR_code							
COD	MCBPC	CBPY	DQUANT	MVD	MVD <sub>2</sub>	MVD <sub>3</sub>	MVD <sub>4</sub>
CR	a0_color	VLC_binary	RLB/ULB	CODA	CBPA	Alpha.Block Data	Block Data

Figure B.2(a) Structure of macroblock layer in I- and P-VOPs

The macroblock layer structure in B-VOPs (VOP\_prediction\_type == '10') is shown in Figure B.2(b). If COD indicates skipped (COD == '1') for a MB in the most recently decoded I- or P-VOP then the co-located MB in B-VOP is also skipped (no information is included in the bitstream). Otherwise, the macroblock layer is as shown in Figure B.2(b).

first_MMR_code							
MODB	MBTYPE	CBPB	DQUANT	MVD <sub>f</sub>	MVD <sub>b</sub>	MVDB	CR
a0_color	VLC_binary	RLB/ULBN	COD	MODBA	CBPBA	A.Block Data	Block Data

Figure B.2(b) Structure of macroblock layer in B VOPs

MODB is present for every coded (non-skipped) macroblock in B-VOP. MVD's (MVD<sub>f</sub>, MVD<sub>b</sub>, or MVDB) and CBPB are present if indicated by MODB. Macroblock type is indicated by MBTYPE which signals motion vector modes (MVD's) and quantization (DQUANT).



### B.1.1 Coded macroblock indication (COD) (1 bit)

A bit which when set to "0" signals that the macroblock is coded. If set to "1", no further information is transmitted for this macroblock; in that case the decoder shall treat the macroblock as a 'P' macroblock with motion vector for the whole block equal to zero and with no coefficient data. COD is only present in VOPs for which VOP\_prediction\_type indicates 'P', for each macroblock in these VOPs.

### B.1.2 Macroblock type & Coded block pattern for chrominance (MCBPC) (Variable length)

A variable length codeword giving information about the macroblock type and the coded block pattern for chrominance. The codewords for MCBPC are given in Table B.1 and Table B.2. MCBPC is always included in coded macroblocks.

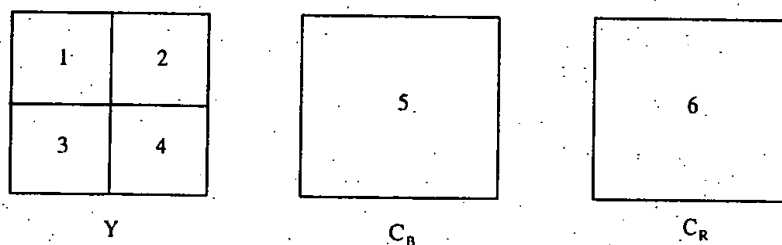
An extra codeword is available in the tables for bit stuffing. This codeword should be discarded by decoders.

The macroblock type gives information about the macroblock and which data elements are present. Macroblock types and included elements are listed in Table B.3 and Table B.4.

Table B.1  
VLC table for MCBPC (for I-VOPs)

Index	MB type	CBPC (56)	Number of bits	Code
0	3	00	1	1
1	3	01	3	001
2	3	10	3	010
3	3	11	3	011
4	4	00	4	0001
5	4	01	6	0000 01
6	4	10	6	0000 10
7	4	11	6	0000 11
8	Stuffing	--	9	0000 0000 1

The coded block pattern for chrominance signifies  $C_B$  and/or  $C_R$  blocks when at least one non-INTRADC transform coefficient is transmitted (INTRADC is the dc-coefficient for 'I' blocks).  $CBPC_N = 1$  if any non-INTRADC coefficient is present for block N, else 0, for  $CBPC_3$  and  $CBPC_6$  in the coded block pattern. Block numbering is given in Figure B.3. When MCBPC=Stuffing, the remaining part of the macroblock layer is skipped. In this case, the preceding COD=0 is not related to any coded or not-coded macroblock and therefore the macroblock number is not incremented. For P-VOPs, multiple stuffings are accomplished by multiple sets of COD=0 and MCBPC=Stuffing.



**Figure B.3**

**Arrangement of blocks in a macroblock**

**Table B.2**  
VLC table for MCBPC (for P-VOPs)

Index	MB type	CBPC (56)	Number of bits	Code
0	0	00	1	1
1	0	01	4	0011
2	0	10	4	0010
3	0	11	6	0001 01
4	1	00	3	011
5	1	01	7	0000 111
6	1	10	7	0000 110
7	1	11	9	0000 0010 1
8	2	00	3	010
9	2	01	7	0000 101
10	2	10	7	0000 100
11	2	11	8	0000 0101
12	3	00	5	0001 1
13	3	01	8	0000 0100

14	3	10	8	0000 0011
15	3	11	7	0000 011
16	4	00	6	0001 00
17	4	01	9	0000 0010 0
18	4	10	9	0000 0001 1
19	4	11	9	0000 0001 0
20	Stuffing	--	9	0000 0000 1

**Table B.3**

Macroblock types and included data elements for a VOP

VOP type	MB type	Name	COD	MCBPC	CBPY	DQUANT	MVD	MVD <sub>24</sub>
P	not coded	-	X					
P	0	INTER	X	X	X		X	
P	1	INTER+Q	X	X	X	X	X	
P	2	INTER4V	X	X	X		X	X
P	3	INTRA	X	X	X			
P	4	INTRA+Q	X	X	X	X		
P	stuffing	-	X	X				
I	3	INTRA		X	X			
I	4	INTRA+Q		X	X	X		
I	stuffing	-		X				

Note: "x" means that the item is present in the macroblock

#### **B.1.4 Coded block pattern for luminance (CBPY) (Variable length)**

Variable length codeword giving a pattern number signifying those Y blocks in the macroblock for which at least one non-INTRADC transform coefficient is transmitted (INTRADC is the dc-coefficient for INTRA blocks).

$CBPY_N = 1$  if any non-INTRADC coefficient is present for block N, else 0, for each bit  $CBPY_N$  in the coded block pattern. Block numbering is given in Figure B.3, the utmost left bit of CBY corresponding with block number 1. For a certain pattern  $CBPY_N$ , different codewords are used for INTER and INTRA macroblocks as defined in Table B.5.

### **B.1.5 Quantizer Information (DQUANT) (1 or 2 bits)**

A one or two bit code to define change in VOP\_quantizer. In Table B.4A and B the differential values for the different codewords are given. VOP\_quantizer ranges from 1 to 31; if the value for VOP\_quantizer after adding the differential value is less than 1 or greater than 31, it is clipped to 1 and 31 respectively. Note that the DQUANT can take values, -2, 0, or 2 in the case of B-VOPs and these values are coded differently from other VOP types.

**Table B.4A**

DQUANT codes and differential values for VOP\_quantizer

Index	Differential value	DQUANT
0	-1	00
1	-2	01
2	1	10
3	2	11

**Table B.4B**

DQUANT codes and differential values for VOP\_quantizer for B-VOPs

Index	Differential value	DQUANT
0	-2	10
1	0	0
2	2	11

**Table B.5**  
VLC table for CBPY

Index	CBPY(I) (12 34)	CBPY(P) (12 34)	Number of bits	Code
0	00 00	11 11	4	0011
1	00 01	11 10	5	0010 1
2	00 10	11 01	5	0010 0
3	00 11	11 00	4	1001
4	01 00	10 11	5	0001 1
5	01 01	10 10	4	0111
6	01 10	10 01	6	0000 10
7	01 11	10 00	4	1011
8	10 00	01 11	5	0001 0
9	10 01	01 10	6	0000 11
10	10 10	01 01	4	0101
11	10 11	01 00	4	1010
12	11 00	00 11	4	0100
13	11 01	00 10	4	1000
14	11 10	00 01	4	0110
15	11 11	00 00	2	11

### ***B.1.6 Motion Vector Coding***

Motion vectors for predicted and interpolated VOPs are coded differentially within a row of macroblocks, obeying the following rules:

In P-VOPs, differential motion vectors are generated as per section on differential coding of Motion vectors (Sec. 3.3.2.6). However, the differential motion vectors are coded as in this section.

- In B-VOPs, every forward or backward motion vector is coded relative to the last vector of the same type. Each component of the vector is coded independently, the horizontal component first and then the vertical component. The prediction motion vector is set to zero in the macroblocks at the start of a row of macroblocks. If a previous macroblock is skipped prediction for current macroblock is reset to zero.
- In B-VOPs, only vectors that are used for the selected prediction mode (MB type) are coded. Only vectors that have been coded are used as prediction motion vectors.

The VLC used to encode the differential motion vector data depends upon the range of the vectors. The maximum range that can be represented is determined by the **forward\_f\_code** and **backward\_f\_code** encoded in the VOP header.

The differential motion vector component is calculated. Its range is compared with the values given in Table B.6 and is reduced to fall in the correct range by the following algorithm:

```
if (diff_vector < -range)
    diff_vector = diff_vector + 2*range;
else if (diff_vector > range-1)
    diff_vector = diff_vector - 2*range;
```

**Table B.6 Range for motion vectors**

forward_f_code or backward_f_code	Range in half sample units
1	32
2	64
3	128

This value is scaled and coded in two parts by concatenating a VLC found from table B.7 and a fixed length part according to the following algorithm:

Let `f_code` be either the `forward_f_code` or `backward_f_code` as appropriate, and `diff_vector` be the differential motion vector reduced to the correct range.

```

if (diff_vector == 0) {
    residual = 0;
    vlc_code_magnitude = 0;
}
else {
    scale_factor = 1 << (f_code - 1);
    residual = (abs(diff_vector) - 1) % scale_factor;
    vlc_code_magnitude = (abs(diff_vector) - residual) / scale_factor;
    if (scale_factor != 1)
        vlc_code_magnitude += 1;
}

```

`vlc_code_magnitude` and the sign of `diff_vector` are encoded according to Table B.7. The residual is encoded as a fixed length code using  $(f\_code - 1)$  bits. If `f_code` is 1 or if `diff_vector` is 0 then the residual is not coded.

### B.1.7 Motion vector data (MVD) (Variable length)

MVD is included for all INTER macroblocks and consists of a variable length codeword for the horizontal component followed by a variable length codeword for the vertical component. Variable length codes are given in Table B.7.

Table B.7  
VLC table for MVD

Index	Vector differences	Bit number	Codes
0	-16	13	0000 0000 0010 1
1	-15.5	13	0000 0000 0011 1
2	-15	12	0000 0000 0101
3	-14.5	12	0000 0000 0111
4	-14	12	0000 0000 1001
5	-13.5	12	0000 0000 1011
6	-13	12	0000 0000 1101
7	-12.5	12	0000 0000 1111
8	-12	11	0000 0001 001
9	-11.5	11	0000 0001 011
10	-11	11	0000 0001 101
11	-10.5	11	0000 0001 111
12	-10	11	0000 0010 001
13	-9.5	11	0000 0010 011
14	-9	11	0000 0010 101
15	-8.5	11	0000 0010 111
16	-8	11	0000 0011 001
17	-7.5	11	0000 0011 011
18	-7	11	0000 0011 101
19	-6.5	11	0000 0011 111
20	-6	11	0000 0100 001
21	-5.5	11	0000 0100 011
22	-5	10	0000 0100 11
23	-4.5	10	0000 0101 01
24	-4	10	0000 0101 11
25	-3.5	8	0000 0111
26	-3	8	0000 1001
27	-2.5	8	0000 1011
28	-2	7	0000 111

29	-1.5	5	0001 1
30	-1	4	0011
31	-0.5	3	011
32	0	1	1
33	0.5	3	010
34	1	4	0010
35	1.5	5	0001 0
36	2	7	0000 110
37	2.5	8	0000 1010
38	3	8	0000 1000
39	3.5	8	0000 0110
40	4	10	0000 0101 10
41	4.5	10	0000 0101 00
42	5	10	0000 0100 10
43	5.5	11	0000 0100 010
44	6	11	0000 0100 000
45	6.5	11	0000 0011 110
46	7	11	0000 0011 100
47	7.5	11	0000 0011 010
48	8	11	0000 0011 000
49	8.5	11	0000 0010 110
50	9	11	0000 0010 100
51	9.5	11	0000 0010 010
52	10	11	0000 0010 000
53	10.5	11	0000 0001 110
54	11	11	0000 0001 100
55	11.5	11	0000 0001 010
56	12	11	0000 0001 000
57	12.5	12	0000 0000 1110
58	13	12	0000 0000 1100
59	13.5	12	0000 0000 1010
60	14	12	0000 0000 1000
61	14.5	12	0000 0000 0110
62	15	12	0000 0000 0100
63	15.5	13	0000 0000 0011 0
64	16	13	0000 0000 0010 0

#### **B.1.8 Motion vector data ( $MVD_{2,4}$ ) (Variable length)**

The three codewords  $MVD_{2,4}$  are included if indicated by  $VOP\_prediction\_type$  and by MCBPC, and consist each of a variable length codeword for the horizontal component followed by a variable length codeword for the vertical component of each vector. Variable length codes are given in Table B.7.

#### **B.1.9 Macroblock mode for B-blocks (MODB) (Variable length)**

MODB is present only in coded macroblocks belonging to B-VOPs. The meaning of this codeword is same as that in H.263. It is a variable length codeword indicating whether MBTYPE and/or CBPB information is present. In case MBTYPE does not exist the default is set to "Direct (H.263 B)". The codewords for MODB are defined in Table B.8.



Table B.8 VLC table for MODB

Index	CBPB	MBTYPE	Number of bits	Code
0			1	0
1		X	2	10
2	X	X	2	11

Note: "x" means that the item is present in the macroblock

#### B.1.10 Macroblock Type (MBTYPE) for Coded B-VOPs (Variable length)

MBTYPE is present only in coded macroblocks belonging to B-VOPs. Furthermore, it is present only in those macroblocks where at least one MVD is sent. MBTYPE indicates the type of macroblock coding used, for example, H.263 like motion compensation or MPEG-1 like motion compensation with forward, backward or interpolated, and change of quantizer if any by use of DQUANT. The codewords for MBTYPE are defined in Table B.9.

Table B.9 MBTYPES and included data elements in coded macroblocks in B-VOPs

Index	MBTYPE	DQUANT	MVD <sub>1</sub>	MVD <sub>2</sub>	MVDB	Number of bits	Code
0	Direct (H.263 B)				X	1	1
1	Interpolate MC + Q	X	X	X		2	01
2	Backward MC + Q	X		X		3	001
3	Forward MC + Q	X	X			4	0001

Note: "x" means that the item is present in the macroblock

Rather than refer to each MBTYPE by an index or by its long explanation in terms of MC mode and Quantizer information, we refer to them as a coding mode which means the following.

- Direct Coding (Direct MC, no new Q)
- Bidirectional Coding (Interpolate MC + Q)
- Backward Coding (Backward MC + Q)
- Forward Coding (Forward MC + Q)

#### B.1.11 Coded block pattern for B-blocks (CBPB) (6 bits)

CBPB is only present in B-VOPs if indicated by MODB.  $CBPB_N = 1$  if any coefficient is present for B-block N, else 0, for each bit  $CBPB_N$  in the coded block pattern. The numbering of blocks has been shown earlier, the utmost left bit of CBPB corresponds to block number 1. When MODB = 0 or 1, the default value of CBPB is set to 0 which means that no coefficients are sent.

#### B.1.12 Quantizer Information for B-Macroblocks (DQUANT) (2 bits)

The meaning of DQUANT and the codewords employed are the same as that in I- or P-VOPs. The computed quantizer is scaled by a factor depending on the selected global quantizer scale for B-VOP's, DBQUANT.

### ***B.1.13 Motion vector data for Forward Prediction ( $MVD_f$ ) (Variable length)***

$MVD_f$  is the motion vector of a macroblock in B-VOP with respect to temporally previous reference VOP (an I- or a P-VOP). It consists of a variable length codeword for the horizontal component followed by a variable length codeword for the vertical component. The variable length codes employed are the same ones as used for MVD and  $MVD_{2-4}$  for P-VOPs.

### ***B.1.14 Motion vector data for Backward Prediction ( $MVD_b$ ) (Variable length)***

$MVD_b$  is the motion vector of a macroblock in B-VOP with respect to temporally following reference VOP (an I- or a P-VOP). It consists of a variable length codeword for the horizontal component followed by a variable length codeword for the vertical component. The variable length codes employed are the same ones as used for MVD and  $MVD_{2-4}$  for P-VOPs.

### ***B.1.15 Motion vector data for Direct Prediction ( $MVDB$ ) (Variable length)***

$MVDB$  is only present in B-VOPs mode if indicated by MODB and MBTYPE and consists of a variable length codeword for the horizontal component followed by a variable length codeword for the vertical component of each vector.  $MVDB$ s represents delta vectors that are used to correct B-VOP macroblock motion vectors which are obtained by scaling P-VOP macroblock motion vectors. The variable length codes employed are the same ones as used for MVD and  $MVD_{2-4}$  for P-VOPs.

## **B.2. Block Layer**

A macroblock structure comprises of four luminance blocks and one of each of the two colour difference blocks. The same structure is used for all types of VOPs, I, P or B. Presently intra macroblocks are supported both in I- and P-VOPs. For such macroblocks, INTRADC is present for every block of each macroblock and TCOEF is present if indicated by MCBPC or CBPY. For nonintra macroblocks of P-VOPs, TCOEF is present if indicated by MCBPC or CBPY. For B-VOP macroblocks, TCOEF is present if indicated by CBPB. Figure B.4 shows a generalized block layer for all type of VOPs.

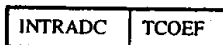


Figure B.4 Structure of block layer

### ***B.2.1 DC coefficient for INTRA blocks (INTRADC) (8 bits)***

A codeword of 8 bits. The code 0000 0000 is not used. The code 1000 0000 is not used, the reconstruction level of 1024 being coded as 1111 1111 (see Table B.10).

Table B.10  
Reconstruction levels for INTRA-mode DC coefficient

Index	FLC	Reconstruction level into inverse transform
0	0000 0001 (1)	8
1	0000 0010 (2)	16
2	0000 0011 (3)	24
.	.	.
.	.	.
126	0111 1111 (127)	1016

127	1111 1111 (255)	1024
128	1000 0001 (129)	1032
252	1111 1101 (253)	2024
253	1111 1110 (254)	2032

### B.2.2 Transform coefficient (TCOEF) (Variable length)

The most commonly occurring EVENTS are coded with the variable length codes given in Table B.11. The last bit "s" denotes the sign of the level, "0" for positive and "1" for negative.

An EVENT is a combination of a last non-zero coefficient indication (LAST; "0": there are more nonzero coefficients in this block, "1": this is the last nonzero coefficient in this block), the number of successive zeros preceding the coded coefficient (RUN), and the non-zero value of the coded coefficient (LEVEL).

The remaining combinations of (LAST, RUN, LEVEL) are coded with a 22 bit word consisting of 7 bits ESCAPE, 1 bit LAST, 6 bits RUN and 8 bits LEVEL. Use of this 22-bit word for encoding the combinations listed in Table B.11 is not prohibited. For the 8-bit word for LEVEL, the codes 0000 0000 and 1000 0000 are not used. The codes for RUN and for LEVEL are given in Table B.12.

Table B.11  
VLC table for TCOEF

INDEX	LAST	RUN	LEVEL	BITS	VLC CODE
0	0	0	1	3	10s
1	0	0	2	5	1111 s
2	0	0	3	7	0101 01s
3	0	0	4	8	0010 111s
4	0	0	5	9	0001 1111 s
5	0	0	6	10	0001 0010 1s
6	0	0	7	10	0001 0010 0s
7	0	0	8	11	0000 1000 01s
8	0	0	9	11	0000 1000 00s
9	0	0	10	12	0000 0000 111s
10	0	0	11	12	0000 0000 110s
11	0	0	12	12	0000 0100 000s
12	0	1	1	4	110s
13	0	1	2	7	0101 00s
14	0	1	3	9	0001 1110 s
15	0	1	4	11	0000 0011 11s
16	0	1	5	12	0000 0100 001s
17	0	1	6	13	0000 0101 0000s
18	0	2	1	5	1110 s
19	0	2	2	9	0001 1101 s
20	0	2	3	11	0000 0011 10s
21	0	2	4	13	0000 0101 0001s
22	0	3	1	6	0110 1s
23	0	3	2	10	0001 0001 1s
24	0	3	3	11	0000 0011 01s
58	1	0	1	5	0111 s
59	1	0	2	10	0000 1100 1s
60	1	0	3	12	0000 0000 101s
61	1	1	1	7	0011 11s
62	1	1	2	12	0000 0000 100s
63	1	2	1	7	0011 10s
64	1	3	1	7	0011 01s
65	1	4	1	7	0011 00s
66	1	5	1	8	0010 011s
67	1	6	1	8	0010 010s
68	1	7	1	8	0010 001s
69	1	8	1	8	0010 000s
70	1	9	1	9	0001 1010 s
71	1	10	1	9	0001 1001 s
72	1	11	1	9	0001 1000 s
73	1	12	1	9	0001 0111 s
74	1	13	1	9	0001 0110 s
75	1	14	1	9	0001 0101 s
76	1	15	1	9	0001 0100 s
77	1	16	1	9	0001 0011 s
78	1	17	1	10	0000 1100 0s
79	1	18	1	10	0000 1011 1s
80	1	19	1	10	0000 1011 0s
81	1	20	1	10	0000 1010 1s
82	1	21	1	10	0000 1010 0s

25	0	4	1	6	0110 0s
26	0	4	2	10	0001 0001 0s
27	0	4	3	13	0000 0101 0010s
28	0	5	1	6	0101 1s
29	0	5	2	11	0000 0011 00s
30	0	5	3	13	0000 0101 0011s
31	0	6	1	7	0100 11s
32	0	6	2	11	0000 0010 11s
33	0	6	3	13	0000 0101 0100s
34	0	7	1	7	0100 10s
35	0	7	2	11	0000 0010 10s
36	0	8	1	7	0100 01s
37	0	8	2	11	0000 0010 01s
38	0	9	1	7	0100 00s
39	0	9	2	11	0000 0010 00s
40	0	10	1	8	0010 110s
41	0	10	2	13	0000 0101 0101s
42	0	11	1	8	0010 101s
43	0	12	1	8	0010 100s
44	0	13	1	9	0001 1100 s
45	0	14	1	9	0001 1011 s
46	0	15	1	10	0001 0000 1s
47	0	16	1	10	0001 0000 0s
48	0	17	1	10	0000 1111 1s
49	0	18	1	10	0000 1111 0s
50	0	19	1	10	0000 1110 1s
51	0	20	1	10	0000 1110 0s
52	0	21	1	10	0000 1101 1s
53	0	22	1	10	0000 1101 0s
54	0	23	1	12	0000 0100 010s
55	0	24	1	12	0000 0100 011s
56	0	25	1	13	0000 0101 0110s
57	0	26	1	13	0000 0101 0111s

83	1	22	1	10	0000 1001 1s
84	1	23	1	10	0000 1001 0s
85	1	24	1	10	0000 1000 1s
86	1	25	1	11	0000 0001 11s
87	1	26	1	11	0000 0001 10s
88	1	27	1	11	0000 0001 01s
89	1	28	1	11	0000 0001 00s
90	1	29	1	12	0000 0100 100s
91	1	30	1	12	0000 0100 101s
92	1	31	1	12	0000 0100 110s
93	1	32	1	12	0000 0100 111s
94	1	33	1	13	0000 0101 1000s
95	1	34	1	13	0000 0101 1001s
96	1	35	1	13	0000 0101 1010s
97	1	36	1	13	0000 0101 1011s
98	1	37	1	13	0000 0101 1100s
99	1	38	1	13	0000 0101 1101s
100	1	39	1	13	0000 0101 1110s
101	1	40	1	13	0000 0101 1111s
102	ESCAPE			7	0000 011

Table B.12  
FLC table for RUNS and LEVELS

Index	Run	Code
0	0	000 000
1	1	000 001
2	2	000 010
.	.	.
.	.	.
63	63	111 111

Index	Level	Code
-	-128	FORBIDDEN
0	-127	1000 0001
.	.	.
125	-2	1111 1110
126	-1	1111 1111
-	0	FORBIDDEN
127	1	0000 0001
128	2	0000 0010

253	127	0111 1111
-----	-----	-----------

## Appendix B

### Core Experiments

The following is a summary list of core experiments. Core experiments have been divided into 9 classes. For more information, please refer to the appropriate documents as listed below.

**Table 1 List of Core Experiments**

No.	Core Experiment
	<b><u>Prediction (ISO/IEC JTC1/SC29/WG11/N1250)</u></b>
P1	Core Experiment on Global Motion Compensation
P2	Core Experiments of Block-Partitioning for Motion Prediction
P3	Core Experiments of STFM/LTFM Memory for Motion Prediction
P4	Motion Segmentation and Compensation for Improved Coding Efficiency
P5	Comparison of Entropy Constrained Variable Block Size Motion Estimation Compensation
P6	2D Triangle Mesh Based MC Prediction
P7	Core Experiment on New Block ME/MC
P8	Core Experiment on Motion and Aliasing Compensating Prediction
	<b><u>Frame Texture Coding (ISO/IEC JTC1/SC29/WG11/N1250)</u></b>
T1/2	Wavelet Coding of I And P Pictures
T3	Matching Pursuits Coding of Prediction Errors
T4	3D-DCT Coding of B-Pictures
T5	Vector Wavelet Coding of I-Pictures
T6	Vector Wavelet Coding of P-Pictures
T7	Core Experiment Description of Modulated Lapped Transform Enhancement Coding
T8	Core Experiment on Variable-Size Lapped Transforms Coding of I & P Pictures
T9/10	Core Experiment on Improved Intra Coding
T11	Adaptive Transform/Quadtree Template Coding of Intra Macroblocks
T12	Residue suppression for frame-based DPCM Coding: Coding of Human Color Perception.
	<b><u>Quantization and Rate Control (ISO/IEC JTC1/SC29/WG11/N1250)</u></b>
Q2	Improved Rate Control
Q3	Analysis of Arithmetic Coding for the MPEG4 video VM

	<b><u>Shape and Alpha Channel Coding (ISO/IEC JTC1/SC29/WG11/N1230)</u></b>
S1	Comparison of Gray Scale Shape Coding Techniques
S2	Geometrical Transformation and Representation of Sprites
S3	Variable Block Size Segmentation
S4	Comparison of Shape coding Techniques
S5	Shape Adaptive Region Partitioning Method for Arbitrary Shaped VOP
	<b><u>Object/Region Texture Coding (ISO/IEC JTC1/SC29/WG11/N1225)</u></b>
O1	Comparison of coding techniques for Sprite objects
O2	DCT coding of macroblocks padded using Alpha-channel
O3	Wavelet/Subband coding of Region Texture
O4	Shape Adaptive DCT for coding of Region Texture
	<b><u>Region Texture Coding (ISO/IEC JTC1/SC29/WG11/N1225)</u></b>
O6	Chroma-keying and block DCT for coding of Region Texture
O7	Mean Replacement DCT coding of Region Texture
O8	E/I (Extension Interpolation) DCT for Region Texture Coding
O9	Coding of Arbitrarily Shaped Textured Image Segments
	<b><u>Error Resilient Core Experiments (ISO/IEC JTC1/SC29/WG11/N1224)</u></b>
E1	Error Resilient Core Experiments on Resynchronization Techniques
E2	Error Resilient Core Experiments on Hierarchical Structures
E3	Core Experiment on Error Resilient Tools
E4	Core Experiment on Error Resilient Methods Based on Back Channel Signaling and FEC
E5	Core Experiment on Error Concealment Techniques
	<b><u>Bandwidth and Complexity Scaling (ISO/IEC JTC1/SC29/WG11/N1266)</u></b>
B1	Generalized Temporal-Spatial Scalable Coding
C1	Content based temporal scalability
	<b><u>Multi-view, Model Manipulation and (ISO/IEC JTC1/SC29/WG11/N1266)</u></b>
M1	Mismatch Corrected Stereo/Multiview Coding
M2	2-D Triangle Mesh for Object/Content Manipulation
	<b><u>Pre-, Mid- and Post-processing (ISO/IEC JTC1/SC29/WG11/N1266)</u></b>
N1	Comparison of Coding Noise Removal Techniques
N2	Comparison of Automatic Segmentation Techniques
N3	Automatic on line generation of Sprites

**THIS PAGE BLANK (USPTO)**